UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE GRADO

El Desarrollo de Software Open Source Analizado desde Dentro

Autor: Carlos López Bartolomé
Tutora: Silvia Teresita Acuña Castillo

Madrid, Julio 2014

Agradecimientos

Gracias a todos los que me habéis ayudado y apoyado estos años. A mis compañeros, que tantas horas hemos pasado juntos, en especial a Goncho, por hacerme divertidas hasta las épocas de exámenes.

Gracias a Silvia Teresita por su dedicación, su compromiso y su inestimable ayuda.

Y por supuesto a mis padres y a Cris, ya que sin su incondicional apoyo no habría podido llegar hasta aquí.

Resumen

El estudio realizado en este trabajo de fin de grado se enmarca en el campo de la Ingeniería del Software. Más concretamente, el proyecto se encuadra en el análisis de las actividades y métodos de trabajo del proceso de desarrollo de *open source software* (OSS) y sus comunidades de usuarios. El objetivo de este trabajo es analizar mediante la participación en proyectos *open source* las actividades que se realizan y compararlas con las llevadas a cabo en los procesos de desarrollo tradicional. Esto significa estudiar los medios de comunicación existentes entre los miembros de las comunidades, sus motivaciones, los métodos de trabajo, qué documentos se desarrollan y qué herramientas se utilizan.

Para ello, se ha realizado un estudio inicial a fin de determinar la situación actual del *software open source* y el proceso de desarrollo de software tradicional. Tras esto se ha participado en un desarrollo tradicional para poder realizar una comparativa práctica en ambos modelos. Esta participación se ha llevado a cabo en la empresa consultora Accenture y se ha colaborado en diferentes actividades de desarrollo de un sistema de *Call Center,* que nos han permitido comprobar que los estudios teóricos existentes son válidos para comprender el funcionamiento del modelo tradicional.

El estudio del OSS se ha llevado a cabo mediante la participación en cinco proyectos de diferentes comunidades, realizando nuevas funcionalidades, corrigiendo errores y aportando mejoras, con el fin de obtener una idea más fiable del comportamiento desde dentro de los desarrolladores de OSS. Los proyectos OSS en los que se ha colaborado son: Double Commander, FileZilla, VLC Media Player, GanttProject y Social Network Manager. El análisis se ha realizado por actividades. La primera actividad estudiada es la de exploración de conceptos, que describe los métodos utilizados en los cinco casos para realizar el primer contacto con la comunidad y las herramientas utilizadas para ello. También se describen las propuestas realizadas y las comunicaciones con la comunidad en las que se exponía si la propuesta era aceptada o rechazada. El siguiente análisis se realiza sobre las actividades de análisis de requisitos, diseño e implementación en los dos casos en los que se ha aprobado nuestra implementación. En estas actividades se explican los métodos que se han seguido para llevar a cabo el desarrollo. Por último se estudian las actividades de mantenimiento y pruebas que, tras la participación en las actividades anteriores, se definen sin analizar los casos de forma individual, sino en forma general, ya que las pruebas son transversales y el mantenimiento engloba a todas las actividades de la mayoría de los casos en los que se ha participado.

Por una parte, los resultados obtenidos muestran una mayor rigidez en el software tradicional, con una gran cantidad de documentación estandarizada, actividades con tareas muy definidas y con una gran preocupación por el factor económico. Por otra

parte, el OSS muestra una evolución basada en la colaboración de los usuarios de las comunidades, sin utilizar prácticamente documentación o estándar de desarrollo y basándose en las directrices de los administradores y las opiniones de los usuarios de las aplicaciones para valorar la validez de las aportaciones propuestas.

Las conclusiones obtenidas en este trabajo permiten conocer los métodos de trabajo de los usuarios de una comunidad *open source*, así como de las herramientas que se utilizan. Se describen además las diferencias reales encontradas entre los dos modelos estudiados tras realizar una participación activa en ellos. Este estudio también abre dos líneas de trabajo basadas en el análisis de las comunidades OSS estudiadas desde el punto de vista de los administradores y la creación de un modelo mixto que extraiga las ventajas encontradas en ambos modelos.

Palabras Clave: Open Source Software, Proceso Software, Desarrollo de Software Tradicional, Actividades Software, Comunidad Open Source.

Abstract

The research and development reported in this final-year project addresses the field of software engineering. To be precise, the project analyzes the activities and working methods used in the open source software (OSS) development process and by OSS user communities. The aim of this project is to conduct a participatory analysis of open source project activities compared with traditional development process activities. This means studying existing forms of communication between community members, motivations, work methods, developed documents and tools used.

To do this, we have conducted a pilot study in order to determine the current state of the OSS and traditional software development processes. We then took part in a traditional development in order to compare practice according to both models. Through cooperation with the Accenture consulting firm, we participated in several activities of the call center system development and verified that existing theoretical studies are valid for understanding the workings of the traditional model.

The OSS study was carried out by participating in five projects across different communities, providing new features, fixing bugs and adding improvements in order to gain a more reliable picture of the behavior of OSS developers from within. The OSS projects in which we have worked include: Double Commander, FileZilla, VLC Media Player, GanttProject and Social Network Manager. The analysis was performed on an activity basis. The first activity is the exploration study of concepts, describing the methods and tools used in all five cases to make contact with the community. The proposals and communications with the community stating whether or not the proposal was accepted are also described. The next analysis is concerned with the activities of requirements analysis, design and implementation where our implementation was approved. The methods applied to carry out the development as part of these activities are explained. Finally, generally applicable maintenance activities and tests are defined without examining individual cases. These activities are discussed, as is cross-testing, and maintenance encompasses all activities in most of the projects in which we participated.

On the one hand, the results for traditional software are less flexible with a lot of very standardized documentation activities and well-defined tasks, showing a lot of concern for the economic factor. On the other hand, the OSS evolves based on user collaboration within communities, using virtually no documents or development standards and following the guidelines established by managers and user reviews of applications to assess the validity of input proposals.

Findings from this study provide insight into the working methods of open source community users, as well as the tools that they use. Further we describe the real differences found between the two models studied after active participation in projects conforming to both models. This study also opens up two lines of research based on the analysis of OSS communities studied from the viewpoint of managers and the creation of a mixed model combining the identified strengths of the two models.

Keywords: Open Source Software, Software Process, Traditional Software Development, Software Activities, Open Source Community.

Índice

Índice de Tablas

Índice de Figuras

CAPÍTULO 1.

INTRODUCCIÓN

El *open source software* (OSS) es un movimiento que ha recibido una enorme atención en los últimos años. Esta nueva forma de realizar software es considerada un auténtico reto para el software comercial, que domina actualmente el mercado de la Informática. El OSS se ha convertido en un componente informático clave en las tecnologías de la información, tanto en la parte puramente de negocio como en sectores específicos (por ejemplo, el educativo), de hecho es un motor que mueve una industria con un valor de miles de millones para compañías como RedHat, Novel e IBM [Rooney, 2005]. Esta creciente importancia del OSS ha propiciado la existencia de estudios que tratan de analizar las diferencias que existen entre el proceso de desarrollo OSS y el proceso convencional de la Ingeniería del Software. Estos estudios cubren todos los aspectos relativos al desarrollo, tales como la motivación para su realización y la participación de los desarrolladores [Hars y Ou, 2002], la forma en que se gestionan los proyectos [O'Mahony, 2003] o los repositorios de código fuente [Mockus et al., 2002].

Existe un gran número de trabajos que describen el proceso de desarrollo que se ha seguido en algunos de los proyectos OSS más populares y exitosos. Por ejemplo, [Mockus et al., 2002] analiza el proyecto Apache, con el fin de obtener una serie de hipótesis que posteriormente confirma estudiando el proyecto Mozilla. El problema de analizar el *open source* de esta forma, es que no existe un modelo de proceso de desarrollo de OSS mundialmente aceptado, que sirva de patrón y defina cómo el OSS se desarrolla en la práctica [Scacchi, 2004; Scacchi, 2005]. Una descripción del proceso es importante para coordinar todas las actividades de desarrollo de software, incluyendo las personas y la tecnología. La coordinación se lleva a cabo dando a los ingenieros de proceso la oportunidad de discutir y administrar las dependencias entre las personas, procesos, actividades y tecnologías en conjunto [Petersen et al., 2008]. La comprensión del contexto, estructura y actividades de los procesos de desarrollo de OSS que se encuentran en la práctica ha sido y sigue siendo un problema difícil de resolver [Scacchi, 2005; Acuña et al., 2012].

Otros trabajos se refieren a estudios basados en un *systematic mapping study* para investigar las actividades de los procesos de OSS [Acuña et al., 2012]. En estos estudios se realiza una búsqueda, dentro de un repositorio de estudios sobre OSS, de términos considerados clave. A partir del análisis de los trabajos seleccionados y clasificados, se obtiene una visión general del problema y se concluye que las actividades de desarrollo de software tradicionales que más aparecen en el proceso de desarrollo promulgado por

las comunidades OSS son: mantenimiento, identificación de ideas o necesidades, educción de requisitos, realización del diseño arquitectónico, implementación del sistema, reporte de errores, realizar revisiones y ejecutar pruebas.

Los estudios mencionados no comparan los procesos OSS desde dentro, es decir, participando personal y directamente en los grupos de actividades de desarrollo en proyectos OSS y estableciendo similitudes y diferencias con los grupos de actividades de proyectos tradicionales. Sin embargo, no hay mejor manera para conocer cómo la comunidad OSS desarrolla software que participar en su proceso de desarrollo realizando un conjunto de aportes [Acuña et al., 2012]. En el proceso de desarrollo de OSS están involucrados diferentes roles (desarrolladores, evaluadores, etc.) desempeñados muchas veces por una misma persona, que realizan sus actividades asociadas.

Este trabajo se centra en conocer y analizar de primera mano el proceso de desarrollo seguido por las comunidades *open source*, ya que como expone [Scacchi, 2004], el primer paso para entender el proceso de desarrollo de las comunidades OSS es investigar cómo las diferentes comunidades desarrollan OSS. Como se ha mencionado, existen numerosos ejemplos de proyectos OSS exitosos, con resultados de alta calidad reconocidos a nivel mundial. Por lo tanto, la motivación de este estudio es que un mayor conocimiento de los procesos de OSS puede informar y mejorar los procesos de desarrollo de software tradicionales. La importancia de este trabajo es comprender qué hace diferente al proceso de desarrollo OSS con respecto al desarrollo más habitual, para determinar qué aspectos del proceso OSS pueden contribuir a mejorar el desarrollo tradicional y así poder sentar las bases de un modelo de desarrollo que contenga los aspectos más positivos y elimine los problemas de los modelos OSS y tradicional.

Por lo tanto, los objetivos de este trabajo son analizar el proceso de desarrollo seguido por proyectos de la comunidad OSS y comparar este proceso con el proceso de software tradicional, mediante la participación directa en ellos.

Para lograr estos objetivos, se seleccionan proyectos de OSS y se vincula a su comunidad de desarrollo, realizando una serie de aportes. En concreto, se participa en un conjunto de actividades desempeñando diferentes roles, como programador, realizando pruebas, aportando nuevas funcionalidades, para conocer cómo es realmente el proceso de desarrollo que siguen las distintas comunidades y analizar las diferencias existentes entre ellas y con el proceso de desarrollo tradicional. Además, se documentará cada una de las actividades realizadas durante el proceso de vinculación con la comunidad y participación dentro del proyecto OSS elegido, desde el primer contacto con las comunidad hasta la finalización de dicha relación.

Una vez se ha conocido como es el proceso de desarrollo de OSS, se deben identificar sus diferencias y similitudes con el proceso de desarrollo tradicional, teniendo en cuenta el conjunto de actividades en las que se ha participado durante el desarrollo. Para ello

utilizaremos un método de investigación llamado estudio de casos en el que cada caso será un proyecto OSS en el que nos adentraremos. El estudio de casos es un método de investigación que se basa en el principio de explorar algo a través de la propia experiencia del investigador (investigación empírica), mediante la observación y participación sin alteración del sujeto u objeto de estudio [Runesson y Höst, 2009]. La información que se trata en este tipo de estudio se basa en las propias experiencias del investigador. Para obtener una visión más realista de las diferencias entre los dos modos de desarrollo considerados, como se ha mencionado, se participará también en un proceso de desarrollo tradicional. Todos los procesos de cada caso serán almacenados en bitácoras (documento en el que se anota todo lo realizado en relación con la comunidad del proyecto considerado, ya sean mensajes intercambiados, actividades realizadas, como opiniones personales).

Este trabajo se ha estructurado del siguiente modo. En el Capítulo 2 se describe el estado de la cuestión analizando los trabajos más relevantes del área de investigación. En el Capítulo 3 se analiza el proceso de desarrollo seguido en el software tradicional, basándose en la teoría y en la experiencia personal adquirida durante la participación en un proyecto de este tipo. En los Capítulos 4, 5 y 6 se explican los diferentes casos estudiados de las comunidades OSS organizados por actividad de desarrollo: exploración de conceptos, análisis de requisitos, diseño e implementación, y pruebas y mantenimiento, respectivamente. En el Capítulo 7 se realiza la comparativa final entre los procesos de desarrollo de la comunidad OSS y los procesos de desarrollo de la Ingeniería del Software tradicional. En el Capítulo 8 se dan las conclusiones de este trabajo de investigación y desarrollo y se describe el trabajo futuro. En los Anexos A y B se especifican estándares internacionales sobre documentación de requisitos y calidad de organizaciones de software. En los Anexos C, D, E, F, G y H podemos encontrar un análisis más detallado de cada caso de estudio.

CAPÍTULO 2.

ESTADO DE LA CUESTIÓN

En este capítulo se describe el conocimiento que existe sobre el estado de la Ingeniería del Software en una comunidad OSS. Los aspectos analizados sobre el proceso de desarrollo, al finalizar nuestra investigación, serán comparados y contrastados con nuestras propias experiencias, comprobando si la conclusión final coincide con la reportada en la literatura.

Existen numerosos trabajos tanto sobre el desarrollo de proyectos OSS como de la forma de desarrollo tradicional. No obstante, para su análisis nos basaremos en la estructura del trabajo de [Potdar y Chan, 2004] por su completitud para comparar el desarrollo de código abierto con el código cerrado; citando actividad por actividad cómo se hace en una corriente y cómo se hace en la otra.

En adelante nos referiremos al proceso de desarrollo de proyectos OSS como PDOSS y al proceso de desarrollo tradicional de la Ingeniería del Software como PDT.

Antes de comenzar con esta comparación, es importante conocer las diferencias que existen entre los equipos de trabajo tanto en PDOSS como en PDT. Por norma general, el PDT es realizado por equipos de trabajo que componen la plantilla de la empresa desarrolladora. Estos empleados realizan las actividades de los distintos procesos del proyecto como parte de su trabajo, siendo gestionadas por la empresa en función de las necesidades del cliente o del objetivo previsto. Esto es importante ya que la motivación que lleva al éxito del proyecto es normalmente económica, pero la motivación personal de los desarrolladores o de las personas implicadas es distinta, y puede incluso que el proyecto en el que participan no sea de su agrado o interés. En los equipos OSS los implicados, tanto desarrolladores como probadores y demás participantes, se vinculan a proyectos que les interesan y normalmente son conocedores del ámbito en el que se desarrolla. Otra característica importante es que al desarrollar pueden incluso elegir proyectos por el lenguaje de programación que les resulte más cómodo o que más conozcan. La motivación personal es total, ya que en la mayoría de los casos los participantes son a su vez, los usuarios finales del producto.

Respecto a los modelos de ciclo de vida del software utilizados, lo más común en el PDT son los modelos en cascada iterativos e incrementales mientras que en el PDOSS lo más frecuente es utilizar un modelo evolutivo, donde el software va evolucionando según las necesidades de los clientes/usuarios [Potdar y Chan, 2004].

A continuación, se describen las actividades comunes en ambos tipos de procesos de desarrollo de software.

2.1. Exploración de Conceptos

Un proyecto PDT comienza con una necesidad, que se va a satisfacer mediante la creación de un software. Esta necesidad puede ser requerida por un cliente que solicita y paga un servicio o porque se ve el desarrollo de ese software como una inversión, ya sea en forma de venta o de beneficios derivados. Una comunidad OSS se construye una vez que la primera versión estable de una aplicación es liberada por sus creadores (uno o dos desarrolladores) y es instalada por los usuarios, quienes reportan errores o solicitan nuevas funcionalidades [Scacchi, 2001]. Como se puede observar, ya en la creación existe una gran diferencia entre estos dos procesos de desarrollo.

2.2. Análisis de Requisitos

Respecto al análisis y especificación de requisitos hay bastante que tratar. En el PDT lo "primero" que se hace es la educción y análisis de requisitos. Estos requisitos no son muy claros al principio y es necesario tener entrevistas con los *stakeholders* (los implicados en el proyecto/el cliente) para aclarar poco a poco lo que quieren de verdad y distinguirlo de lo que realmente necesitan, para empezar con el diseño y la implementación del software [Satzinger et al., 2000]. En estas entrevistas y para obtener la información y aprobación del usuario se utilizan un gran número de técnicas, tales como las maquetas, prototipos, escenarios, etc. que permiten obtener un conjunto de requisitos adecuado. En el PDOSS son los propios usuarios (obviamente los desarrolladores también tienen voz aquí pues muchas veces son los mismos) los que piden y proponen las funcionalidades de las que se caracterizará el sistema, ya que son las personas que utilizan la aplicación y son los que mejor pueden dar ideas útiles sobre su funcionamiento.

A la hora de discutir los requisitos a implementar, varían los participantes. En el PDOSS participan un amplio número de personas/desarrolladores (suelen ser usuarios del sistema también) mientras que en el PDT están muy diferenciados: los ingenieros del software y los *stakeholders*. Algo que tienen en común los dos procesos es que ambos negocian los requisitos que entran en conflicto. Los requisitos en el PDT son contractuales, lo que implica que han de llevarse a cabo para que se considere por terminado el proyecto y por lo tanto están menos sometidos a cambios durante la vida del mismo. Esto es diferente en el PDOSS. Los requisitos se estipulan de forma escrita, sobre la marcha, mediante correos electrónicos, listas de correo o sitios web. Los requisitos no suelen estar documentados en este último proceso o bien si lo están, de forma menos formalizada. Según un estudio realizado por [Acuña et al., 2012] la mayoría de los proyectos OSS estudiados no incluyen actividades relacionadas con la

validación de requisitos lo cual quiere decir que en el PDOSS no se realiza la validación de la misma manera que en el PDT, lo cual es lógico, pues en el PDOSS no se busca la realización de unos requisitos fijos, sino cumplir con las necesidades que los usuarios van reportando.

2.3. Diseño

De manera resumida, podemos decir que en relación con el diseño en el PDT se dedica mucho tiempo en pensar cuál va a ser el diseño del sistema mientras que en el PDOSS muchas veces se mezcla este proceso con el de la codificación. Según [Acuña et al., 2012] solo dos actividades del PDT están presentes en el PDOSS: la realización del diseño de la arquitectura y la realización del diseño detallado (esto deja fuera al diseño de la base de datos y al diseño de las interfaces). El diseño de la arquitectura del sistema software del PDOSS está implícito y va evolucionando con el paso del tiempo [Vixie, 1999]. Otro aspecto en el que se diferencian las dos corrientes es en el peso que tienen el diseño y la implementación. Por un lado, en el PDOSS el diseño limpio y elegante no es una práctica común. El diseño es secundario y la implementación cobra mayor importancia [Johnson, 2001]. Por otro lado, en el PDT la implementación tiene la misma importancia que el diseño [Acuña et al., 2012]. El motivo de esta diferencia, como se ha mencionado anteriormente, es que normalmente el PDT desarrolla software para vender, por lo que el diseño es un aspecto fundamental, ya que para la mayoría de usuarios es el principal referente de calidad.

En cuanto al nivel de implicación de cada persona en la comunidad existen varias diferencias. En el PDOSS los usuarios pueden contribuir y participar en el diseño del proyecto vía *mailing lists* [Senyard y Michlmayr, 2004; Mockus et al., 2002]. En el PDT los usuarios no intervienen en esas actividades de diseño y son los desarrolladores los que lo hacen bajo unas definidas asignaciones de trabajo.

2.4. Implementación

La implementación es una de las actividades más importantes en ambos procesos. En el PDOSS los equipos de desarrollo están distribuidos y ellos mismos pueden elegir qué quieren implementar (dependiendo de sus gustos y preferencias) lo que a priori tiene cierta incidencia en la calidad del código. Justo al contrario, en el PDT, en general los equipos de desarrollo están centralizados y los desarrolladores reciben los productos de software que tienen que implementar desde la línea jerárquica más vertical, ya sean de su agrado o no. En el PDOSS el código está abierto a todo el mundo por lo que cualquier usuario puede contribuir en esta tarea [Schweik y Semenov, 2003; Dinh-Trong y Bieman, 2005]. Por el contrario, en el PDT sólo los desarrolladores tienen derecho a escribir código y éste solo está disponible para los miembros del equipo.

Respectivamente, en el PDT sólo los desarrolladores están capacitados para aportar código en esta actividad [Senyard y Michlmayr, 2004] mientras que en el otro caso todos los usuarios técnicos pueden hacerlo.

Por último y como ya hemos mencionado, debemos resaltar que en el PDOSS, la codificación es una forma de documentación [Reis y Mattos Fortes, 2002]. Cuando se escribe código, los usuarios suelen aportar comentarios de qué están haciendo y para qué sirve.

2.5. Pruebas

En la actividad de pruebas, también denominada *testing*, podemos encontrar que en el PDOSS los usuarios reportan errores y actúan como *beta testers* [Potdar y Chan, 2004]. Cuando un usuario encuentra un error puede o bien solucionarlo o bien reportárselo a la comunidad [Webber, 2000] para que otro usuario lo solucione. El *testing* en OSS puede ser considerado como una actividad clave, pues mediante las pruebas de los usuarios es como se logra mejorar el software. En el PDT las pruebas, tanto las que se realizan durante el desarrollo como las finales, suelen definirse al mismo tiempo que se educen los requisitos. Otra característica es que las pruebas las pueden llevar a cabo los propios desarrolladores, los usuarios finales e incluso un tercero contratado para ello. Una vez que se ha puesto en funcionamiento el software, los errores que se vayan descubriendo serán corregidos mediante *service packs*.

Una característica muy importante del PDOSS es la frecuencia con la que las *releases* se llevan a cabo (una vez al día o a la semana) mientras que en el PDT, estas *releases* se dan una vez al año o incluso con menos frecuencia [Potdar y Chan, 2004].

2.6. Documentación

En cuanto a la documentación podemos encontrar diferencias entre el PDOSS y el PDT. En el PDT se documenta todo o casi todo lo que se hace ya sean el plan de gestión del proyecto software, la especificación de requisitos del software o los procedimientos de diseño y pruebas. En el PDOSS nada o casi nada se documenta de forma oficial. La forma de documentar en este último proceso más frecuente es en el propio código o en foros de discusión o similares [Reis y Mattos Fortes, 2002].

2.7. Mantenimiento

El aspecto que posiblemente marque más la diferencia entre estas dos formas de desarrollar software es el mantenimiento. En el PDT el mantenimiento puede dividirse en dos tipos: el mantenimiento postentrega, que describe el mantenimiento como

cualquier cambio que se realice sobre un software ya entregado e instalado [IEEE Std. 1219:1998]; y el mantenimiento moderno, que son las actividades correctivas, perfectivas o de adaptación que se realizan en cualquier momento [ISO/IEC 12207:1995], siendo el postentrega un subconjunto del mantenimiento moderno. En cualquiera de los dos casos, es importante remarcar que el PDT es el modo utilizado de forma mayoritaria para el desarrollo de software a nivel de negocio, por lo que todos los aspectos relacionados con el mantenimiento, como el alcance, el número de revisiones o la adaptación a nuevas tecnologías, están descritos en el contrato del proyecto. El PDOSS se construye a partir una serie de colaboraciones o aportes sobre una idea o un proyecto inicial, por lo que no contiene un grupo de actividades de mantenimiento como el PDT.

Se puede considerar que las actividades que se realizan en el proceso de mantenimiento de forma tradicional, surgen como aportes de los miembros de la comunidad, que buscan mejorar la calidad del software que se está desarrollando, por lo que también se podría decir que los PDOSS siempre están, mientras haya gente colaborando en ellos, en proceso de mantenimiento.

2.8. Resumen de Diferencias entre el PDT y el PDOSS

Las diferencias que existen sobre el PDT y el PDOSS se basan en que el primero consta de un ambiente centralizado mientras que en el segundo gobierna más la colaboración entre los usuarios, conformando un entorno descentralizado [Potdar y Chan, 2004]. En la Tabla 2.1 se muestran las diferencias entre el PDT y el PDOSS analizadas.

Una idea planteada por [Scacchi, 2001] afirma que la manera de trabajar del PDOSS es más barata, más rápida y de mayor calidad que la del PDT ya que, partiendo del principio de que los usuarios del PDOSS trabajan en un proyecto porque quieren (la motivación juega un papel importante aquí), no cobran nada por el trabajo (el coste del proyecto se reduce), trabajan en paralelo y motivados ya que hacen algo que han seleccionado (esto hace que se trabaje más rápidamente) y publican el trabajo una vez está finalizado de verdad, no tienen fechas programadas (por lo que la calidad del proyecto aumenta considerablemente). En el PDT es todo lo contrario (aunque sus trabajadores, de igual modo, pueden estar motivados). La seguridad del PDT y el PDOSS es algo subjetivo ya que no por ser de código abierto va a ser más o menos seguro que por ser de código cerrado. La idea de seguridad se centra más en una idea o filosofía. En el PDT la seguridad se consigue mediante la oscuridad. Esconder el código es una manera de esconder las brechas de seguridad (lo que no quiere decir que sea infranqueable). En el PDOSS la seguridad se consigue enseñando el código. Al tener todo el mundo acceso al código, éste puede ser accedido las veces que haga falta y ser modificado y mejorado con el paso del tiempo [Potdar y Chan, 2004].

Actividad	Características del PDT	Características del PDOSS	Diferencias
Exploración de Conceptos	- Evaluación de la situación problemática existente - Definición de los objetivos preliminares a alcanzar - Estudio de la factibilidad técnica y económica - Busca un beneficio económico	- Busca satisfacer una necesidad - La viabilidad es decidida por los usuarios de la comunidad, que determinan que es correcto para la comunidad - Altruista	En el PDT se analiza de forma muy detallada la viabilidad, ya que se deben estimar unos costes. En el OSS no existe dicho análisis y depende básicamente de la opinión de los usuarios.
Análisis y Especificación de Requisitos	- Educción de requisitos - Requisitos contractuales - Roles específicos y poco numerosos participan en las tareas de decisión - Existen actividades de validación de requisitos	- Los usuarios piden nuevas funcionalidades - Requisitos estipulados de forma escrita sobre la marcha - Gran cantidad de personas participan a la hora de decidir - No existen actividades de validación de requisitos	Podemos observar que en el PDT todo está más centralizado (la "autoridad" reside en unos pocos). Además se rige por un grado más alto de formalidad que en el PDOSS.
Diseño	- Se dedica tiempo a pensar en el diseño - El diseño tiene el mismo peso que la implementación - Sólo los desarrolladores participan en el diseño y bajo unas definidas asignaciones de trabajo	- No se piensa mucho en el diseño. Se mezcla con la actividad de implementación - El diseño tiene menos peso que la implementación - Tanto los desarrolladores como lo usuarios participan en el diseño	En el PDOSS el diseño se incorpora dentro de la implementación mientras que en el PDT se siguen una serie de procesos definidos y asignados cuidadosamente para que el diseño sea claro y facilite la implementación más adelante.
Implementación	- Equipos de desarrollo centralizados - Áreas asignadas a los equipos - Sólo los desarrolladores acceden al código	- Equipos de desarrollo distribuidos - Los usuarios eligen qué implementar - Todo el mundo puede acceder al código	En OSS el colaborador decide cómo realizar la implementación y el grado de implicación en hacerla más comprensible para los demás desarrolladores. En el PDT es vital una buena documentación.
Pruebas	- Uso de *service packs* para solucionar los errores - Planes de prueba previamente estipulados	- Los usuarios reportan los errores encontrados y en algunos casos también los solucionan	En el PDOSS se confía más en el criterio del usuario que usa la aplicación

Tabla 2.1: Características y Diferencias entre PDT y PDOSS por Actividad

Actividad	Características del PDT	Características del PDOSS	Diferencias
Mantenimiento	- En el PDT, la identificación de las mejoras se realiza sobre la base de diferentes documentos, resultado de otras actividades relacionadas con la planificación del proyecto - En el desarrollo tradicional, los usuarios solo reportan errores, no los corrigen	- Los miembros del equipo núcleo de desarrolladores son los que deciden si incorporan o no una nueva funcionalidad - Cualquier persona, en cualquier momento, puede sugerir o aportar mejoras - Los usuarios finales OSS que actúan como desarrolladores o como encargados de realizar el mantenimiento producen continuamente estas mejoras - Los reportes de problemas o solicitudes de mejoras pueden ser realizados por cualquier persona, incluidos los usuarios	El proceso de mantenimiento tradicional del software no encaja con lo que ocurre en la comunidad OSS. En su lugar, puede ser mejor caracterizar la dinámica general de la evolución OSS como reinvención. Esta reinvención emerge continuamente de la adaptación, aprendizaje, y mejora de las funcionalidades y calidad del OSS.
Documentación	- Se documenta todo o casi todo de lo que se hace	- La propia documentación reside en el código o en los foros	En el PDT se documenta mediante un procedimiento más formalizado.

Tabla 2.1: Características y Diferencias entre PDT y PDOSS por Actividad (Continuación)

CAPÍTULO 3.

PROCESO DE DESARROLLO TRADICIONAL

En este capítulo se explican las actividades por las que atraviesa un proyecto software en lo que hemos llamado PDT. En este capítulo se definirán dichas actividades sobre la base de la estructura del proceso software descrito en [IEEE Std. 1074:2006] y se contrastarán con la experiencia adquirida durante la participación en un PDT, lo que más adelante nos permitirá evidenciar las diferencias con el proceso seguido en OSS.

La empresa desarrolladora del proyecto en el que se ha participado es Accenture. Esta empresa es una multinacional dedicada a la prestación de servicios de consultoría, servicios tecnológicos y de *outsourcing* y, por tanto, es un ejemplo adecuado de modelo de proceso de desarrollo y metodologías empleados en los PDT.

El proyecto en el que se ha participado consiste en la adecuación de los sistemas *de Call Center* para nuevas ofertas y productos de una conocida empresa de telecomunicaciones, que llamaremos TelcoO. El departamento con el que se ha trabajado es el de *Software Quality Assurance* (SQA), encargado de probar el software y solicitar las modificaciones necesarias en los diferentes sistemas constituyentes, para garantizar la calidad pactada con el usuario.

Por motivos de privacidad no se va a revelar cierta información relacionada con la empresa y el proyecto implicado, ocultándola en los correos y documentos que se presentan de manera adjunta como ejemplo de desarrollo.

3.1. Exploración de Conceptos y Planificación

En primer lugar se ha de llevar a cabo una definición o un estudio del software que se pretende desarrollar. Esta actividad es la primera y una de las más decisivas de un proyecto. En ella se evalúa la situación problemática existente, se definen los objetivos preliminares a alcanzar y se efectúa el análisis del entorno del proyecto. Además se realiza como parte del proceso de toma de decisiones inicial y primordial en estas acciones iniciales, un estudio de la viabilidad técnica, económica del mercado potencial, y la selección de la alternativa más apropiada.

Dentro de los factores que se analizan, se encuentra la limitación de recursos, el presupuesto o coste del desarrollo del software y por supuesto, la definición del proceso de desarrollo y la estructura de las fases o modelo de proceso a seguir durante el proyecto.

Por último, se elabora un plan de proyecto que servirá para controlar el estado en el que se encuentra y facilitar su posterior evaluación.

3.2. Análisis de Requisitos

Para que se concluya con éxito un PDT, es de crucial importancia que se tenga una completa y plena comprensión de los requisitos del software. La tarea de análisis puede verse como un proceso de descubrimiento, donde se refina en detalle el ámbito del software, creando modelos de requisitos de datos, flujos de información y donde se analizan soluciones alternativas. En resumen permite especificar la funcionalidad, rendimiento y las restricciones que debe cumplir el software.

El análisis de los requisitos puede dividirse en una serie de tareas. En primer lugar hay que reconocer los elementos básicos del problema, evaluando cómo los perciben los usuarios finales. A continuación hay que entender el comportamiento del software en el contexto de acontecimientos que afectan al sistema. Otro paso, que ayuda al usuario a comprender como será el comportamiento externo del software final, es la creación de modelos de sistema tales como maquetas o prototipos. Por último, se lleva a cabo una especificación formalizada de requisitos y una revisión general de todo el proceso de análisis.

La especificación de los requisitos es una tarea fundamental, pues va a servir como prueba para la valoración del resultado final tanto por parte del usuario como del desarrollador. Los requisitos se pueden dividir en los siguientes tipos:

- Requisitos funcionales: derivan de los requisitos de capacidad y especifican la funcionalidad o servicios que la aplicación debe proporcionar. Suelen estar acompañados de un modelo de análisis que engloba los requisitos de información, de operación y los modelos de comportamiento que sean necesarios.

- Requisitos no funcionales: derivan de los requisitos de restricción y su función es la de imponer restricciones en el producto desarrollado, en el proceso de desarrollo además de indicar cómo se debe realizar dicho software.

El proceso de análisis de requisitos requiere diferentes actividades de alto nivel y son desarrollados por múltiples agentes, como usuarios, expertos de dominio, expertos de

marketing, programadores, etc. Para la formulación de las especificaciones existen diferentes estándar, el más conocido es el estándar [IEEE Std. 830:1998]. Dicho estándar fue generado por un equipo de trabajo del IEEE, su finalidad es la integración de los requisitos del sistema desde la perspectiva del usuario, cliente y desarrollador. En el Anexo A se describe el estándar IEEE 830-1998 para una buena documentación de requisitos.

El análisis global de los requisitos de una aplicación es, por tanto, un proceso de conceptualización y formulación de los conceptos que involucra de forma concreta. Es una parte fundamental del proceso de desarrollo de una aplicación, ya que la mayor parte de los defectos encontrados en el software entregado se originan en la actividad de análisis de requisitos, y además son los más caros de reparar durante el desarrollo.

Existe la problemática de entender quién es el responsable de los requisitos, el cliente o el desarrollador. Para gestionar esto, es habitual presentar el análisis de requisitos en dos secciones, una con los requisitos de cliente, documentan los deseos y necesidades de los clientes y se expresan en lenguaje claro para él y otros requisitos de software más detallados, que los determina de manera específica y estructurada, pues están destinados específicamente hacia los desarrolladores.

3.3. Diseño

En el diseño básicamente se discute cómo la lista de requisitos funcionales y no funcionales se traduce en una aplicación software. Se puede decir que una vez planteada la especificación de requisitos de software, hay que analizar desde un punto de vista técnico las posibles soluciones. Entre ellas, se elige la que se considera más adecuada. A partir de ese momento, se decide la estructura general del software, dividiéndolo en componentes y marcando las relaciones entre ellos.

En el análisis de cada uno de los componentes pueden existir varias soluciones, estas decisiones se toman entre los miembros del equipo, consultando al cliente en caso de ser necesario un cambio en el alcance o en alguno de los requisitos.

El diseño suele dividirse en niveles. En el nivel más alto se estructura la arquitectura del software. En el nivel más bajo del diseño hay que decidir la estructura de control y el flujo de datos de cada módulo. El uso de la programación estructurada facilita enormemente la comprensión de los algoritmos, al limitar los flujos de control posibles. El producto final de la etapa de diseño puede ser un organigrama, unas líneas de pseudocódigo, etc.

Todas las decisiones de diseño quedan reflejadas en documentos que servirán de base para la codificación del producto final por parte de los programadores. A continuación, se describen algunos de las herramientas para la elaboración de dichos documentos:

- Diagrama de flujo: es la representación gráfica que más se utiliza en el diseño procedimental. Para representar un paso de procesamiento se utiliza un cuadro, para representar una condición se utiliza un rombo, y para representar el flujo de control se utilizan flechas.

- Diagrama de cajas: Esta notación surgió del deseo de desarrollar una representación para el diseño procedimental que no permitiera la violación de construcciones estructuradas. En ellas el ámbito funcional queda bien definido y de forma claramente visible, siendo sencillo determinar el ámbito de los datos locales y globales y la representación de recursos de codificación como la recursividad.

Existen una serie de estándares para la descripción del diseño de un sistema software, como el estándar [ISO/IEC/IEEE 42010:2011], que se ocupa de las actividades de la creación, el análisis, y el mantenimiento de las arquitecturas de sistemas intensivos en software, y el registro de este tipo de arquitecturas en cuanto a las descripciones de diseño de la arquitectura. Además incorpora anexos que proporcionan el fundamento de los conceptos clave y la terminología, las relaciones con otros estándares y ejemplos de uso.

3.4. Implementación

Esta actividad consiste en llevar a cabo el desarrollo de lo especificado en los requisitos. Si el diseño es adecuado y suficientemente detallado, la codificación de cada módulo se puede considerar algo casi automático.

Una de las principales decisiones a tomar en esta actividad es la del lenguaje de programación a emplear, aunque la mayoría de las veces el diseño ya lo establece de forma implícita.

Evaluar la calidad de la codificación es una tarea difícil. Para un mismo diseño son posibles muchas implementaciones diferentes y no existen criterios claros que permitan decidir cuál es la mejor. En este punto, existen una serie de métricas del software que pueden ser utilizadas para valorar el código realizado.

Es evidente que valorar la calidad es algo bastante complicado, ya que es un concepto diverso, la Real Academia Española define la calidad como la propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor. El estándar de calidad en [ISO 8402:1995] dice que la calidad es la totalidad de rasgos y características de un producto, proceso o servicio que sostiene la habilidad de satisfacer estados o necesidades implícitas. Dentro de las muchas medidas de calidad, éstas son las consideradas más importantes:

- Corrección: un programa debe operar de forma correcta o no aportará ningún valor a sus usuarios. La corrección es el grado en que el software lleva a cabo las funciones requeridas.

- Facilidad de mantenimiento: como se comprobará más adelante, el mantenimiento del software es la actividad que cuenta con mayor esfuerzo durante el desarrollo de un PDT. La facilidad de mantenimiento mide la facilidad con la que se puede corregir un programa si se encuentra un error, la facilidad de adaptación a nuevas necesidades del cliente o de optimización si se desea un cambio de requisitos. Debido a no poder medir directamente esta característica se utilizan medidas indirectas como el tiempo medio de cambio (TMC), es decir, el tiempo que se tarda en analizar la petición de cambio, en diseñar una modificación apropiada, en efectuar el cambio, en probarlo y en distribuir el cambio a todos los usuarios.

- Integridad: debido al aumento de los intrusos informáticos y de virus, la integridad es de fundamental importancia. Este atributo mide la habilidad de un sistema para soportar ataques, ya sean intencionados o accidentales, contra su seguridad.

- Usabilidad: mide la facilidad de aprendizaje, la eficiencia y facilidad de uso de la interacción del software con el usuario y se consigue medir en función de cuatro características, que son (1) la destreza intelectual solicitada para aprender el sistema; (2) el tiempo que se requiere para llegar a ser eficiente en el uso del sistema; (3) aumento de la productividad en comparación con el sistema al que reemplaza; y (4) una valoración subjetiva de la disposición u opinión de los usuarios hacia el sistema.

Para la codificación también se definen una serie de estándares de estilo que facilitan la legibilidad y claridad del software producido y ayudan a los programadores a entender el código que no ha sido realizado por ellos, algo de crucial importancia debido a que en la mayoría de los proyectos que siguen el PDT participan un gran número de desarrolladores. La calidad del servicio a proporcionar y del desarrollo, instalación y de las evaluaciones llevadas a cabo en un PDT se deben asegurar en los proyectos de software. En el Anexo B, Estándares de Calidad ISO 9000, se describen las normas involucradas.

3.5. Pruebas

En esta actividad hay que comprobar que las especificaciones aprobadas por el usuario de la aplicación y los desarrolladores se cumplen perfectamente y en todos los casos. Este proceso puede ser llevado a cabo por el usuario que solicita el software, por la empresa desarrolladora o incluso por un tercero. En general, los responsables de las

pruebas deben fiarse del análisis, diseño y código para que les guíen en el diseño y ejecución de los casos de prueba. Esta tarea es crucial y ha de realizarse de forma correcta, pues en un futuro, los errores que se pasen por alto pueden agravarse cuando se realicen nuevas versiones.

Además de las pruebas de programas se utilizan técnicas de verificación y validación, utilizando una aproximación estructurada para analizar y probar el software en relación con todos los aspectos del sistema en el cual se incluye, y en especial con el hardware, los usuarios y las interfaces con otras piezas de software.

Idealmente, la verificación y validación se realiza paralelamente al desarrollo de software, durante todo su ciclo de vida, y pretende alcanzar los siguientes objetivos: descubrir errores de alto riesgo, dando al equipo de diseño la oportunidad de elaborar una solución adecuada; evaluar el ajuste de los productos desarrollados a las especificaciones del sistema; y proporcionar al equipo de gestión información actualizada sobre la calidad y el progreso del esfuerzo de desarrollo.

3.6. Mantenimiento

El tipo de mantenimiento de un sistema software suele quedar definido al inicio del proyecto, indicando las actualizaciones, el periodo entre actualizaciones y la duración del mismo.

Durante el mantenimiento el software, indudablemente, sufrirá cambios y habrá que realizar modificaciones a su funcionalidad. Es de suma importancia que el software pueda adaptarse a los cambios en su entorno. Esas posibles extensiones han de contener la documentación adecuada, que permita presentar nuevas vías para el mantenimiento y modificaciones de las mismas. En un gran número de casos estas modificaciones se suelen presentar como *releases* o versiones del primer producto, que se encuentran en repositorios y que permiten a los usuarios del software acceder a nuevas funcionalidades o a funcionalidades antiguas que ya no contengan errores.

Para asegurar que el mantenimiento se realiza de forma correcta, el estándar [IEEE 982.1:1988] sugiere un índice de madurez del software (IMS) que proporciona una indicación de la estabilidad de un producto de software, basada en los cambios que ocurren con cada versión del producto (Figura 3.1).

$$IMS = [Mr - (Fa + Fc + Fd)]/ Mr$$

Siendo:

Mr = número de módulos en la versión actual

Fa = número de módulos en la versión actual que se han añadido

Fc= número de módulos en la versión actual que han cambiado

Fd = número de módulos eliminados de la versión anterior

Figura 3.1: Fórmula Para Calcular el Índice de Madurez del Software (IMS)

A medida que el IMS se aproxima a 1.0 el producto se empieza a estabilizar. El tiempo medio para producir una versión de un producto software puede correlacionarse con el IMS desarrollándose modelos empíricos para el mantenimiento.

Por último hay que tener en cuenta que un PDT puede incluir la instalación de componentes físicos para su utilización. Por supuesto el mantenimiento de dichos equipos, así como otros factores físicos tales como el consumo o el espacio físico que ocupen también deben quedar definidos dentro del mantenimiento.

3.7. Estándares IEEE

Durante la descripción de las actividades por la que atraviesa un PDT, se han mencionado una serie de estándares que aseguran la calidad en la realización y documentación de un proyecto software. Además de los nombrados, existen también estándares para otras actividades relacionadas con la calidad como pruebas, verificación y validación, revisiones, etc. La Tabla 3.1 recoge los principales estándares.

ESTÁNDAR	DESCRIPCIÓN
IEEE 730-2002	Planes de aseguramiento de la calidad del software
IEEE 829-1998	Documentación de pruebas del software
IEEE 982.1, 982.2	Diccionario estándar de medidas para producir software fiable
IEEE 1008-1987	Pruebas de unidad del software
IEEE 1012-1998	Verificación y validación del software
IEEE 1028-1997	Revisiones del software
IEEE 1044-1993	Clasificación estándar para anomalías del software
IEEE 1061-1998	Estándar para una metodología de métricas de calidad del software
IEEE 1228-1994	Planes de seguridad del software

Tabla 3.1: Descripción de los Principales Estándares IEEE

3.8. Análisis del PDT en un Proyecto Real

En este trabajo se ha participado en el desarrollo de nuevas funcionalidades para un proyecto ya realizado y que se encuentra en estado productivo. El motivo de la selección de este proyecto es que permite una comparación directa con los PDOSS, ya que estos normalmente se basan en aportaciones a sistemas ya desarrollados. Toda la información complementaria sobre este proyecto se encuentra en el Anexo C, PDT Real: Accenture Madrid.

El proyecto sobre el que se va a trabajar consiste en el *Call Center* de la compañía TelcoO, en la parte denominada ULL o cobre, es decir, la parte referida a las líneas fijas tradicionales. A partir de ahora nos referiremos al *Call Center* como CC. La nueva funcionalidad que se va a desarrollar se basa en la creación de un producto adicional que se podrá incorporar a una serie de ofertas de telefonía. Este producto se denomina 5 números plus y se va a utilizar su proceso de desarrollo para comprender de forma más clara las actividades de un PDT.

En primer lugar aparece el análisis conceptual, que lógicamente surge de la necesidad, por motivos de negocio, del cliente de incluir este producto adicional en el sistema CC. Para ello se inicia la negociación con la empresa desarrolladora Accenture. Al finalizar dicho proceso de negociación se elabora un documento, Anexo C, cuya finalidad queda descrita en el apartado propósito, como se puede comprobar en la Figura 3.2.

*El objetivo de la presente petición es ajustar dicha propuesta comercial para cubrir las necesidades en cuanto a las llamadas desde el fijo y hacia el fijo dentro del ámbito de cliente S*** y M****.*

En este SO se definen las tareas a realizar en el proyecto 'Diferenciación Pro 5 números plus gratuitos', para que las necesidades del Usuario sean operativas en el entorno de producción.

Queda cerrado en este documento con la aprobación del usuario:

Requisitos a implementar en producción acordados con el Usuario (pueden no coincidir con los del PO).

- *Solución a implementar.*
- *Desarrollo realizar por los proveedores.*
- *Soporte a dar por los proveedores.*
- *Recursos necesarios para desarrollar la solución.*
  - *Entregas por parte de Desarrollo.*
  - *Entregas por parte del Usuario.*
- *Pruebas a realizar.*
- *Despliegue a realizar.*
- *Coste de la solución detallado por cada uno de los proveedores.*
- *Planificación.*

Figura 3.2: Documento Análisis Conceptual: Apartado Propósito

Una vez definidos estos criterios se realizan las actividades de educción de requisitos, diseño y desarrollo. Estas actividades se van a agrupar ya que debido al carácter del software que se está desarrollando, se utilizan los requisitos directamente para programar la funcionalidad, pues en este caso, Accenture solo programa un sistema, siendo tarea de los responsables del resto de sistemas elaborar los diseños y codificaciones necesarias para el correcto funcionamiento. La Tabla 3.2 se obtiene del documento de especificación de requisitos, Anexo C, y muestra cómo quedan definidos algunos de los requisitos.

ID-RU	Nombre	Descripción	Fecha	Área	Prioridad
1.	*Ampliación a 5 números Plus*	*Cualquiera de las ofertas vigentes para captación de ADSL + Llamadas o fibra Empresas dispondrá de forma gratuita e irá asociado por defecto 10 números plus gratuitos.* *Estará disponible solo para:* • *Altas de ADSL +Llamadas del portfolio vigente en captación:* o *ABPLU* o *ABPET* o *ABSET* o *AB4MT* o *AB1MT* • *Altas de Fibra +Llamadas del portfolio vigente en captación:* o *ABFPU*	*17/07 /2013*	*MKT UNE*	*Alta*
2.	*Contratación*	*En el caso de altas, en las herramientas de venta aparecerá marcado por defecto sin que el comercial tenga que marcar.* *Estará disponible también para contratar en postventa para clientes de cartera que se dieran de alta con anterioridad a este desarrollo y que se encuentren en las ofertas listadas con anterioridad. En cuyo caso deberán seleccionar la promo.*	*17/07 /2013*	*MKT UNE*	*Alta*
5	*Cambios de oferta*	*En los cambios de oferta hacia las ofertas destino listadas con anterioridad la promoción irá asociada por defecto.*	*17/07 /2013*	*MKT UNE*	*Alta*
6	*Ficha de producto*	*Referencia a otro documento.*			

Tabla 3.2: Especificación de Requisitos

Cuando el software está desarrollado, se pasa al equipo de pruebas, en este caso SQA, que es donde se ha participado. Para la realización de las pruebas, se elabora un plan de pruebas referido al producto adicional 5 números plus, Anexo C. En este documento se enumeran todos los casos de prueba que el cliente va a validar, junto con una descripción detallada de cada uno de los casos. Los testers deberán ejecutar dichas pruebas y obtener las evidencias necesarias para que el usuario las valide. Para facilitar la comprensión de la ejecución de las pruebas se va a analizar el proceso de validación de un único caso de prueba. La Tabla 3.3 muestra la definición de dicho caso en el plan de pruebas.

#	CASO	Descripción	Detalles	Estado
1	PA_5NP_GRATIS_EMPRESA - Alta de cliente con oferta Par Vacante ABPLU	Lanzar a través de Xnet un Alta de cliente con oferta Par Vacante ABPLU contratando el nuevo PA Promo 5 NP gratis	Comprobar que al dar un alta en Xnet de Par Vacante ABPLU, al llegar a la pantalla de los productos adicionales no se ofrece el antiguo PADE43 sino el nuevo PA Promoción 5 Números Plus gratis (Empresas) y que sigue apareciendo el PABT02 (Números Plus Empresa) de forma opcional y desmarcado. Comprobar que el nuevo PA Promoción 5 Números Plus gratis, aparece obligatorio y marcado por defecto.	No Run

Tabla 3.3: Definición del Caso 1 del Plan de Pruebas

Para ejecutar esta prueba se procede a dar un alta de la oferta solicitada ABPLU en el sistema Xnet. Al llegar a la ventana en la que se seleccionan los productos adicionales, se observa un error, ya que como muestra la Figura 3.3, el antiguo 3 números plus se sigue ofreciendo de forma obligatoria y el nuevo 5 números plus aparece como opcional:

Figura 3.3: Página de Contratación de Productos Adicionales (*Call Center*)

Una vez detectado un fallo, se reporta al sistema responsable para que lo solucionen, enviando un correo electrónico, Correo Electrónico C.1. En él se especifica el fallo y se

adjuntan las evidencias necesarias. Tras revisarlo, el equipo de desarrollo nos responde, Correo Electrónico C.2, informándonos que el documento de requisitos no coincide con lo que se pide en la prueba y explicándonos que para cambiarlo necesita un cambio en los requisitos. A continuación se ha de reportar esta incidencia al cliente, ya que o bien se cambia el plan de pruebas o se cambia el alcance del proyecto, Correo Electrónico C.3.

Debido a que este proceso es largo, se realizan informes diarios que se envían por correo a los jefes de equipo, lo que les permite llevar un control de todos los miembros del equipo. En el informe del día en que se descubre este error, se explica el bloqueo total que se está sufriendo, Correo Electrónico C.4.

A partir de este momento, se producen una serie de reuniones con el cliente para decidir la solución más adecuada. En concreto se decide cambiar el alcance y ajustar los requisitos a lo que se especifica en la prueba. Este cambio se hace oficial en un correo, Correo Electrónico C.5, donde se especifican los cambios a realizar y los tiempos de desarrollo de cada uno de los equipos implicados.

Tras finalizar el desarrollo, se vuelve a realizar la prueba, en este caso de manera satisfactoria. Para que el cliente acepte y de por buena la prueba, se ha de elaborar un documento en el que se aportan la evidencias necesarias para que el usuario pueda comprobar que todo está correcto y considere como buena la prueba.

El documento de evidencias se sube a un sistema en el que se encuentra el plan de pruebas del proyecto y donde pueden acceder tanto el equipo de pruebas como el cliente. La prueba se sube con la evidencia y en el estado pendiente de validación. El usuario accede al sistema y cambia el estado a *passed* si la prueba es correcta, como muestra la Figura 3.4.

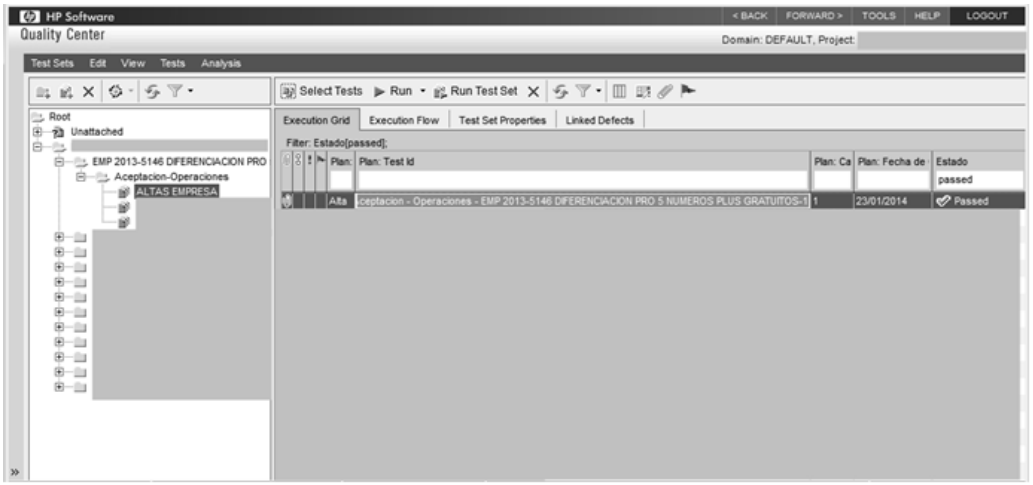


Figura 3.4: Sistema de Validación de Pruebas: Caso 1

CAPÍTULO 4.

EXPLORACIÓN DE CONCEPTOS EN PROYECTOS OSS

En el grupo de actividades denominado exploración de conceptos se incluyen las diferentes formas mediante las cuales se puede establecer contacto o iniciar la participación con una comunidad de OSS. Se pretende dar una visión real de cómo se puede llevar a cabo una colaboración con los desarrolladores de OSS. Dentro de las cuestiones de la exploración de conceptos que se quieren abordar están la búsqueda, comprensión y alcance de proyectos, así como el conocimiento de las diferentes aplicaciones y sus funcionalidades, lo que nos permitirá proponer nuevas funcionalidades, participar mediante la corrección de errores y la elaboración de componentes adaptativos de la interfaz de usuario y de funcionalidades que mejoren la usabilidad de la aplicación.

A continuación, informaremos sobre los casos estudiados.

4.1. Planteamiento del Problema

Como en otros casos de estudio que se analizarán en próximos capítulos, nos surge la pregunta de cómo se llevan a cabo las diferentes actividades de los proyectos OSS. En concreto, este capítulo se centra en un análisis inicial, o de toma de contacto con diferentes comunidades. A lo largo del análisis de estas tareas no se va a profundizar en los métodos de desarrollo de las comunidades OSS, sino que se recogerán una serie de experiencias, que permitan conocer todos los medios disponibles para iniciarse en proyectos OSS.

Para ello, como usuario común se ha intentado entrar en contacto con diferentes comunidades, buscando participar activamente en ellas, ya sea proponiendo mejoras e incluso el desarrollo de estas propuestas contribuyendo con aportaciones, ayudando a otros colaboradores, reportando errores o colaborando en otras funciones que se requieran. Así nos introduciremos dentro de varios proyectos y podremos conocer cómo se trabaja, lo que será de vital importancia para el análisis de las siguientes actividades.

4.2. Objetivos de Investigación

El objetivo de este capítulo es comprender de forma práctica, como una persona con interés en OSS puede llevar a cabo una colaboración en un proyecto de este modelo de desarrollo. Para ello se describirán diferentes casos, tanto exitosos como fallidos, en los que se podrán conocer los métodos para acceder a las comunidades OSS.

Otro de los objetivos de esta investigación es obtener experiencia que permita comparar las formas y motivaciones de la participación en proyectos OSS frente a las de los PDT.

4.3. Diseño del Estudio sobre la Actividad de Exploración de Conceptos

En este apartado describiremos los procedimientos a seguir para llevar a cabo el estudio de los casos. Estos procedimientos son:

- Realizar un conjunto de cuestiones cuyo fin es el de responderlas mediante la participación en las diferentes comunidades de OSS.
- Establecer un protocolo de análisis de los datos que obtenemos durante la participación.
- Establecer unos criterios de validación que deben cumplirse para que el caso sea admitido como válido.

4.3.1. Preguntas de Investigación

¿Cómo se establece contacto con la comunidad? ¿Cómo educen los requisitos? ¿Quién propone las nuevas funcionalidades? ¿Se pueden apoyar las propuestas de otros usuarios? ¿Qué plataformas utilizan para reportar incidencias o propuestas? ¿Qué sistema de comunicación tienen?

4.3.2. Procedimiento de Análisis

En primer lugar se va a analizar cómo se ha tomado contacto con la comunidad y la respuesta que recibimos, quién nos contesta y qué nos comunica. Se contemplará la posibilidad de que no se establezca contacto alguno, si no se recibe respuesta.

También se analizarán situaciones en las que no se establezca un contacto directo, sino que simplemente se proponga una nueva funcionalidad, usando las diferentes maneras que cada comunidad tiene de hacerlo. Así también observaremos cómo se comunican propuestas y cuánto difieren los medios de comunicación entre las comunidades de desarrollo OSS.

Por último se analizará quién y de qué manera nos comunica si se puede implementar o colaborar. Se comprobarán las herramientas utilizadas para estas actividades y se compararán con las utilizadas en los demás casos analizados.

4.3.3. Validación

La validación de los casos se realizará sobre las preguntas de investigación propuestas anteriormente. Si un caso ha sido válido o no es algo subjetivo ya que estamos en la actividad de exploración de conceptos y, en realidad, toda participación activa con una comunidad nos otorgará información útil. Por tanto el éxito de un caso dependerá de la información que nos aporte, buscando obtener la mayor información posible acerca de los diferentes métodos de comunicación y exploración de conceptos en OSS.

4.4. Descripción de los Casos

4.4.1. Identificación del Caso: Caso #1 Double Commander

4.4.1.1. Resumen

DoubleCommander es un gestor de archivos a doble pantalla que permite mover archivos entre directorios de una forma sencilla. Incluye también otras funcionalidades como un editor de texto integrado y un visualizador de archivos en binario, hexadecimal y en formato texto.

Tras descargar la aplicación de su repositorio, se realizó un uso continuado de la misma, para conocer su funcionalidad y poder aportar mejoras o cambios. Durante este proceso se dio con una funcionalidad que podía ser mejorada. Ésta consistía en que mientras se exploran los archivos, hay un botón que deja de mostrar el explorador principal para mostrar otra información. Cuando se desea volver a navegar por los archivos, no se puede volver al sitio exacto donde se había dejado.

Buscando el desarrollo de esta aportación, se inició un contacto con la comunidad para implementar un nuevo botón que, tras acceder a la información, te permitiese volver a la misma ruta por la que estabas navegando. Tras hacer la propuesta, se recibió una contestación de la comunidad en la que se denegaba el desarrollo, ya que existía un comando que implementaba la funcionalidad propuesta.

4.4.1.2. Análisis del Estudio del Caso

Selección del caso: Este proyecto fue elegido por ser una aplicación distinta a las aplicaciones de esta funcionalidad con las que suelo trabajar. De ahí el interés por conocer una nueva forma de realizar el trabajo aprovechando la oportunidad.

Recolección de datos: Se ha desarrollado un documento (bitácora) en el que se detallan todos los pasos realizados desde el primer contacto con la comunidad hasta que se

finalizó la participación o se redactó este documento para los casos en los que aún se mantiene una participación activa con la comunidad. La bitácora abarca la comunicación con la comunidad, información sobre la aplicación, herramientas utilizadas, etc. La bitácora completa sobre la participación en el caso Double Commander se describe en el Anexo D, Bitácora de Double Commander.

4.4.1.3. Resultados

En la página web del proyecto, doublecmd.sourceforge.net, se encuentra toda la información necesaria para poder reportar incidencias, realizar propuestas y así poder contribuir a la mejora de la aplicación. Se utilizó Mantis como medio para proponer nuevas funcionalidades. Mantis es una aplicación que permite a los usuarios reportar errores y realizar propuestas para nuevas funcionalidades. El error o aportación es recibido por los desarrolladores, quienes lo asignan a los miembros de la comunidad que pidan trabajar en dicho error/aportación.

La funcionalidad que se sugirió trataba de solucionar un problema de este explorador de archivos a doble pantalla. El problema consistía en que mientras se exploran los archivos, hay un botón que deja de mostrar el explorador para mostrar otra información y al volver al punto donde se estaban explorando los archivos, no se puede. En un primer momento se pensó que sería pulsando en el mismo botón que abría la información, pero no es así.

La funcionalidad propuesta consistía en implementar un botón que permitiese volver al explorador de archivos, manteniendo el directorio en el que se estaba navegando. Esta propuesta fue denegada, ya que ésta ya estaba implementada de otra manera. La contestación desde la comunidad, Figura 4.1, explicaba que la funcionalidad que se estaba proponiendo ya existía.

> Such “button” already exists. It´s directory history. It can be called by mouse click on path or by assign hotkey to cm_ViewHistory command.

Figura 4.1: Contestación a la Funcionalidad Propuesta en el Caso #1

Como se puede comprobar, Figura 4.1, no se trataba exactamente de lo que se proponía. Actualmente se puede volver al directorio utilizando el historial, ya sea haciendo click en el botón historial y buscando la ruta deseada o usando un comando.

La propuesta realizada buscaba sencillez y rapidez mediante un botón volver. El motivo de no aprobarla posiblemente se encuentre en que ese tipo de aplicación no está orientada a usuarios sin ningún conocimiento técnico, por lo que el uso del comando implica directamente las características propuestas. El perfil de usuario que usa esa aplicación es un usuario con cierto conocimiento técnico por lo que el hecho de que las cosas sean *user-friendly* no se valora mucho.

La persona que denegó el aporte fue un desarrollador, que contaba con derechos de administrador en Mantis. Los usuarios pueden añadir sus comentarios, pero la potestad sobre qué se desarrolla y qué no la tienen los administradores de la aplicación.

4.4.1.4. Conclusiones del Caso

Resultados de la participación: Tras este estudio de caso, se corrobora que cualquier usuario común puede aportar nuevas funcionalidades a una comunidad, pudiendo buscar proyectos dentro de sus intereses. También queda patente que los foros y plataformas similares como los *trackers*, la ya descrita Mantis es un ejemplo de ellos, permiten reportar errores o nuevas funcionalidades de forma directa y sencilla.

Opinión personal: La herramienta estaba bastante bien definida, por lo que no se propuso ninguna nueva funcionalidad para implementar aparte de la ya descrita. Algo que queda muy patente tras esta colaboración es la facilidad de comunicación con la comunidad y sus miembros, incluso tratándose de una aplicación muy completa, lo que muestra el interés que existe en el OSS, en todo momento, por mejorar.

Otro aspecto interesante es que, pese a que todo usuario que lo desee puede proponer funcionalidades o detectar errores, son una serie de administradores los que tienen el poder de decisión. Hay que destacar que esos administradores son los que mejor comprenden la herramienta porque han trabajado directamente en el desarrollo de la aplicación. De esta manera se asegura que la aplicación mantiene las características por las que se desarrolló, labor de los demás usuarios es apoyar las propuestas de los demás para que los desarrolladores con más autoridad vean el apoyo y la aprueben.

4.4.2. Identificación del Caso: Caso #2 FileZilla

4.4.2.1. Resumen

FileZilla es un cliente FTP, FTPS y SFTP que permite cargar o descargar archivos en o desde un servidor. Es una herramienta muy útil para desarrolladores de software ya que, por ejemplo, permite cargar los componentes de un sitio web en su servidor, o descargarlos desde remoto. Filezilla tiene una interfaz de usuario gráfica e intuitiva y es un sistema rápido y confiable.

Esta aplicación es utilizada en el entorno de trabajo, por lo que se conocían muy bien todas sus características. Tras registrarse en su comunidad, se intentó colaborar como usuario de todas las formas posibles, eligiendo en primer lugar añadir un posible cambio en los atajos. Para ello se realizó una petición dentro de la sección de errores. Pasado un tiempo de aproximadamente un mes, se recibe comunicación por correo donde se informa que la petición no está en el sitio correcto y que se ha creado la zona de futuros cambios, adjuntándome el id de la petición. Tras ello se descarga el código para empezar a trabajar, pero al poco tiempo se añade una respuesta donde se asegura haber solucionado el problema, lo cual era cierto.

Durante este periodo, se observa cómo en la comunidad se da gran importancia a las traducciones, llevando un seguimiento exhaustivo del estado de las mismas según el idioma. Como el español no estaba finalizado, se procede a contribuir en la traducción, siguiendo el *workflow* que se describía en la página. Este proceso finaliza de manera exitosa y en futuras *releases* puede comprobarse el resultado.

Por último, se realiza una petición, ésta de forma correcta, en la que se propone añadir la velocidad de transferencia de ficheros, durante la copia o descarga, algo que se echa en falta durante el uso de la aplicación. Pero no se recibe contestación al respecto.

4.4.2.2. Análisis del Estudio del Caso

Selección del caso: El principal motivo por el que se ha decidido trabajar con esta aplicación es por el gran uso que se le da en el entorno de trabajo en el que me encuentro. Aparte del uso personal, los exploradores de archivos, y en concreto FileZilla, son aplicaciones muy utilizadas en el desarrollo de software y en otras actividades, por lo que la posibilidad de realizar una aportación en uno de ellos me parecía muy interesante.

Recolección de datos: Se ha desarrollado un documento (bitácora) en el que se detallan todos los pasos realizados desde el primer contacto con la comunidad, hasta que se finalizó la participación o se redactó este documento para los casos en los que aún se mantiene una participación activa con la comunidad. La bitácora abarca la comunicación con la comunidad, información sobre la aplicación, herramientas utilizadas, etc. La bitácora completa sobre la participación en el caso FileZilla se describe en el Anexo E, Bitácora de FileZilla.

4.4.2.3. Resultados

En la página web de FileZilla Project, https://filezilla-project.org, se puede encontrar toda la funcionalidad necesaria para realizar los distintos tipos de colaboraciones con la comunidad. Además contiene una serie de manuales donde se describe cómo realizar la comunicación con la comunidad, describiendo tanto los medios como las formas. Por ejemplo, se puede encontrar un manual de cómo realizar una petición de cambio o reportar un error y un manual de cómo redactar correctamente un comentario contestando a otro usuario.

La primera aportación a la comunidad consistía en reportar un error que se había descubierto durante el uso de la aplicación. El problema consistía en que cuando se estaba trabajando con archivos locales, si se desea editarlos de forma directa, se puede navegar hasta ellos y pulsando la tecla E iniciar la edición, apareciendo un *popup* de confirmación, que se acepta pulsando ENTER, por lo que la pulsación de E + ENTER se hace de forma casi inmediata. Si por el contrario se está trabajando en remoto, por ejemplo viendo los ficheros de un servidor, al pulsar E se eliminan archivos, abriéndose una ventana de confirmación. Al estar acostumbrado a editar, la primera vez se pulsa E

+ ENTER sin prestar atención y se eliminan archivos. Para reportar este fallo se coloca una petición dentro de la sección Bugs. Pasado un tiempo de aproximadamente quince días se recibe un correo donde se explica que no se trata de un error, sino de un posible cambio, por lo que la petición no está creada de forma correcta. Se adjunta un Id donde se puede acceder a una petición en la sección de propuestas futuras, proponiendo el cambio descrito en el error y asignándole prioridad alta. Esto permite llevar un correcto seguimiento de las aportaciones de los usuarios.

Una vez reportado, se siguen los manuales de instalación de entornos y descarga de ficheros para intentar implementar la solución de este fallo nosotros mismos. Debido al desconocimiento del código, antes de poder dar con la solución se añade una contestación a nuestra propuesta donde se indica el fichero donde se encuentra el cambio y las modificaciones necesarias. Tras revisar el cambio en el código descargado se comprueba que está solucionado. Los administradores cambian el tipo a la petición, pasándola a *patch*, siendo una mejora para las próximas versiones.

La siguiente aportación consiste en colaborar con la traducción de la aplicación. La página del proyecto contiene una sección donde se pueden ver todas las traducciones que se han realizado y el estado en el que se encuentran. También contiene los pasos a seguir para realizar una traducción, permitiendo de forma *online* compilar los ficheros para poder probarlos previamente en la aplicación, en caso de tener descargado el código. Para completar el castellano se descarga el fichero correspondiente y se procede a realizar la traducción. Una vez implementada se usa el compilador *online* para generar un archivo de prueba compilado. Este archivo se puede probar en el código descargado, añadiendo un directorio con el código del lenguaje al que se ha traducido. Una vez se ha comprobado que funciona, se envía la traducción a un correo que se facilita en la página, donde comprueban que es correcto y actualizan la información.

Por último y para crear una petición de forma correcta, se comprobaron todos los manuales e indicaciones y se propuso una futura mejora. Esta mejora consistía en añadir la velocidad del intercambio de ficheros, algo que no aparece y que puede no ser necesario para archivos pequeños, que son los que normalmente se usan, pero que puedes ser muy útil a la hora de subir aplicaciones de gran tamaño a servidores o descargar grandes repositorios para realizar pruebas. Una vez creada la petición de mejora se esperó una respuesta, pero que al día de la redacción de este documento aún no ha llegado.

4.4.2.4. Conclusiones del Caso

Resultados de la participación: El aspecto fundamental que se debe destacar de la participación en FileZilla en la gran cantidad de documentación para garantizar una buena gestión y comunicación con la comunidad. En lo referente al sistema de seguimiento del estado de las traducciones hay que destacar el control total que se lleva,

llegando a indicar incluso los cambios restantes para completar la traducción y mostrando barras de estado de las mismas.

Opinión personal: Todos los medios de colaboración con la comunidad están perfectamente indicados, en poco tiempo se pueden leer los manuales, muy bien realizados y comenzar la colaboración con la comunidad. El registro no es más que un nombre de usuario, un correo y una contraseña, por lo que proponer una mejora o reportar un error es algo rápido y que podría realizar cualquier usuario, esté o no familiarizado con la informática y el OSS. Esto me parece fundamental para la evolución de las aplicaciones y debería ser una característica esencial de todos los proyectos OSS.

4.4.3. Identificación del Caso: Caso #3 VLC Media Player

4.4.3.1. Resumen

VLC Media Player es un reproductor de vídeo y audio muy conocido que reproduce un gran número de formatos de vídeo, audio o subtítulos además de tener la capacidad de reproducir en streaming. Como usuario de la herramienta y, por tanto, conocedor de sus funcionalidades se encontró una utilidad que podía ser añadida, por lo que se inició un contacto con la comunidad para proponerla. Esta funcionalidad consistía en añadir una barra lateral al reproductor, que permita navegar por los ficheros del ordenador y poder seleccionarlos para reproducir al momento o ir colocándolos en la lista de reproducción. La comunidad denegó dicha propuesta, ya que consideraban que estaba implementada en un botón situado en la zona inferior del reproductor.

Al no tratarse exactamente de la misma funcionalidad, se rehízo la propuesta explicando mejor el cambio, pero se obtuvo la misma respuesta.

4.4.3.2. Análisis del Estudio del Caso

Selección del caso: Este proyecto fue elegido ya que VLC Media Player es el reproductor que uso habitualmente y la propuesta realizada pretendía mejorar una tarea que suponía uno de los pocos fallos de la aplicación. Desconocía que se trataba de un programa OSS y al verlo en una de las comunidades en las que se estaba trabajando, se consideró una buena oportunidad para mejorar una aplicación de uso habitual.

Recolección de datos: Se ha desarrollado un documento (bitácora) en el que se detallan todos los pasos realizados desde el primer contacto con la comunidad, hasta que se finalizó la participación o se redactó este documento para los casos en los que aún se mantiene una participación activa con la comunidad. La bitácora abarca la comunicación con la comunidad, información sobre la aplicación, herramientas utilizadas, etc. La bitácora completa sobre la participación en el caso VLC Media Player se describe en el Anexo F, Bitácora de VLC Media Player.

4.4.3.3. Resultados

En su página web, www.videolan.org/vlc/, se puede encontrar numerosa información sobre cómo colaborar con el proyecto, principalmente en las actividades relacionadas con el reporte de errores, por lo que se decidió utilizar la lista de correo para consultar el medio por el que se debían proponer las nuevas funcionalidades. Al poco tiempo se recibió una respuesta afirmando que ese era el medio, por lo que se procedió a realizar la descripción de la nueva funcionalidad que se deseaba añadir. Dicha funcionalidad consistía en la posibilidad de, mientras se está reproduciendo un archivo, se pueda explorar los directorios desde el reproductor, en una barra o menú lateral, para o bien reproducirlos o añadirlos a la cola de reproducción, evitando así tener que hacerlo directamente con el navegador de archivos del sistema operativo usado. El menú o barra debería poderse ocultar cuando no se esté usando para no interferir con la reproducción.

Al día siguiente se recibió la siguiente respuesta, Figura 4.2, de un miembro de la comunidad, subscrito a la lista de correo y de rol desconocido.

> Unless I have gotten you wrong, this already exists in the playlist menu i.e. the playing audio or video resumes into a smaller screen at the bottom left, you see playlist on right and in left pane i.e. above the smaller audio / video, there is file system browse section where you can browse and add any file while the main one keeps running until you play the new one.

Figura 4.2: Primera Respuesta de un Administrador a la Proposición de Mejora en el Caso #3

Como se puede comprobar se nos comunicó que la funcionalidad pedida ya existía, aunque en realidad no era del todo cierto. Simplemente se podía realizar del siguiente modo: si se hace click sobre el botón de *playlists*, se abre un menú donde solo se puede navegar por los archivos situados en las carpetas "Mi Música", "Mi Vídeo" y "Mis Imágenes". Pero esto difiere mucho de la idea propuesta ya que es posible que el usuario sitúe sus archivos en diferentes directorios.

Se supuso que no se había expuesto bien la posible mejora, por lo que se volvió a escribir en la lista de correo explicando de forma más detallada que lo que se había propuesto no era exactamente lo que se hacía referencia en la respuesta de la comunidad. Pese a realizar una propuesta mejor redactada y explicada no se volvió a recibir respuesta.

Tras esto se decidió exponer la propuesta en el foro de la aplicación. Se volvió a recibir la misma respuesta negativa de un desarrollador, Figura 4.3.

> FYI, VLC already has a sidebar where you can navigate files in "My Music/Videos/Pictures".

Figura 4.3: Segunda Respuesta de un Administrador a la Proposición de Mejora en el Caso #3

Se respondió en el foro aclarando la propuesta pero, al igual que en el intento anterior, no se volvió a recibir respuesta.

Se dio por finalizada la participación en el proyecto. Como ya se ha comentado, aparte de los desarrolladores, cualquier usuario puede comentar en los *posts* de los demás, pero la última potestad de decisión la tienen los administradores de la aplicación.

4.4.3.4. Conclusiones del Caso

Resultados de la participación: Tras este estudio de caso, se corrobora que cualquier usuario común puede aportar nuevas funcionalidades a la comunidad. También queda patente que los foros y plataformas similares como listas de correo son el medio de reporte habitual de peticiones de nuevas funcionalidades.

Opinión personal: Algo que se ha podido sacar en claro es que cualquier usuario puede colaborar con las propuestas de los demás usuarios escribiendo en ellas, pero sólo algunos administradores son los que deciden sobre la aprobación de la propuesta (como ya hemos indicado en el apartado de resultados). La labor de los demás usuarios es apoyar las propuestas de los demás para que los desarrolladores con más autoridad vean el apoyo y la aprueben.

El no recibir contestación al mensaje en el que se aclara la propuesta, tras la primera negación, es una muestra de que el excesivo número de propuestas que los proyectos OSS reciben de los usuarios hace que no puedan dedicarle el tiempo necesario a cada propuesta y da la sensación de que la propuesta no ha sido evaluada correctamente. Esto se ve influido por el sistema de listas de correo, donde no quedan tan bien descritas las aportaciones y es casi imposible, debido a la afluencia de mensajes, realizar un seguimiento más allá de un par de días. Esto es algo negativo ya que de propuestas incompletas pueden surgir nuevas funcionalidades muy originales. En resumen, el modo de educir requisitos no es un debate entre usuarios (lo cual aportaría mucha riqueza y calidad a las propuestas) sino que es un "juicio". Algunos desarrolladores encargados de aceptar o denegar propuestas leen la propuesta y dan un veredicto, no aportando mejoras o relacionando propuestas.

4.4.4. Identificación del Caso: Caso #4 GanttProject

4.4.4.1. Resumen

GanttProject es un software de gestión de proyectos basado en Java. Cuenta con la mayoría de las funciones básicas de gestión de proyectos, como puede ser la creación de diagramas de Gantt, programación de tareas y la gestión de recursos, utilizando tablas

de carga de recursos. No contiene otras funciones más específicas que sí tienen otros programas de este tipo, ya que se puede decir que está diseñado teniendo en cuenta el principio KISS, es decir, mantener el software sencillo debido a que así funcionará mejor.

Como no se conocía de manera directa este software, se inició el contacto con la comunidad buscando directamente en la lista de errores reportados por otros usuarios. Cuando se encontró uno que se creía poder solucionar, se solicitó la asignación, recibiendo una respuesta positiva.

4.4.4.2. Análisis del Estudio del Caso

Selección del caso: Este proyecto fue elegido ya que se había trabajado con programas de planificación de proyectos software, pero todos eran bastante complicados de utilizar. Por comentarios de otros compañeros comentando su sencillez, se analizó su página en la comunidad OSS y se vio que existía mucha actividad, por lo que se decidió buscar alguna forma de colaborar en él.

Recolección de datos: Se ha desarrollado un documento (bitácora) en el que se detallan todos los pasos realizados desde el primer contacto con la comunidad, hasta que se finalizó la participación o se redactó este documento para los casos en los que aún se mantiene una participación activa con la comunidad. La bitácora abarca la comunicación con la comunidad, información sobre la aplicación, herramientas utilizadas, etc. La bitácora completa sobre la participación en el caso GanttProject se describe en el Anexo G, Bitácora de GanttProject.

4.4.4.3. Resultados

El sistema de comunicación entre los usuarios de la comunidad de este proyecto es similar al del Caso #2. Una de las diferentes acciones que se pueden realizar una vez registrado es colaborar en los parches que se están desarrollando actualmente, ya sea aportando ideas adicionales, solucionando dudas o errores que aparezcan durante la realización y probando alguno de los cambios. También permite proponer nuevas funcionalidades, que una vez se aprueben y se asignen, ya sea porque los administradores las creen oportunas, o porque los usuarios las demandan, pasarán a formar parte del apartado de parches. Por último se permite reportar errores.

En el apartado de errores, llamado Bugs, se dio con un error en el que se decidió colaborar para tratar de solucionarlo. El error consistía en una serie de *pop-ups* que aparecían en las opciones del menú y que entorpecían la visualización, ya que se superponían. Para solicitar la adjudicación de la solución del error o al menos para informar de la inmediata disposición a ello, se escribió un mensaje en *post* del error.

Al poco tiempo se recibió una contestación de otro usuario dando vía libre para solucionar el error (Figura 4.4).

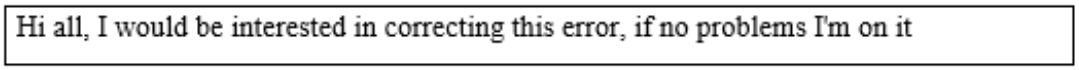


Figura 4.4: Solicitud de Asignación de Mejora en el Caso #4

Como se expondrá en el Capítulo 5 de este documento, se procede con el desarrollo de este error hasta su correcta finalización.

4.4.4.4. Conclusiones del Caso

Resultados de la participación: El interés principal de este caso es corroborar la importancia de los medios que permiten la comunicación entre usuarios. Al igual que en otros casos estudiados, el seguimiento detallado de las aportaciones, ya sean errores o desarrollo y propuestas de nuevas funcionalidades es fundamental para el funcionamiento efectivo de la comunidad OSS.

Opinión personal: Este era el primer proyecto en el que se participaba sin conocer de manera directa la funcionalidad de la aplicación. Esto, al igual que pasa en el ámbito profesional, no tiene por qué impedir la realización de aportaciones ya que con un buen sistema de comunicación y conociendo el código en el que se desarrolla el proyecto es suficiente.

Algo que se ha podido observar es que en muchas comunidades no es posible filtrar proyectos por el lenguaje en el que están desarrollados. Esto puede servir para saber de antemano si se puede o no colaborar en ellos, reduciendo el número de peticiones de desarrollo fallidas.

4.4.5. Identificación del Caso: Caso #5 Colaboración en Proyecto OSS Social Network Manager

4.4.5.1. Resumen

Este caso trata de la colaboración en un desarrollo para un futuro proyecto OSS dedicado a la gestión de redes sociales. La colaboración se obtuvo desde la página openhatch.org, en la que se ofertan colaboraciones en proyectos OSS. Una de las colaboraciones consistía en la implementación de una funcionalidad para medir relaciones entre usuarios de Twitter, con la ventaja de que el lenguaje en el que se debía realizar el desarrollo se conocía muy bien. Se procedió a solicitar la asignación, siendo ésta aceptada. Tras una larga colaboración se finalizó de forma satisfactoria, como se describe en el Capítulo 5.

4.4.5.2. Análisis del Estudio del Caso

Selección del caso: Este proyecto fue elegido ya que podría permitir conocer los pasos previos a la formación del proyecto OSS, algo de lo que no se había tenido oportunidad

en los otros casos, pues formaban parte de proyectos ya desarrollados. También se eligió porque ya se había trabajado con el lenguaje en el que se debía realizar la funcionalidad, lo que facilitaba la colaboración.

Otro de los motivos que permitieron dar con esta oportunidad son el sistema de búsqueda de aportaciones a proyectos OSS por lenguaje de programación, algo que se había echado en falta en otras colaboraciones.

Recolección de datos: Se ha desarrollado un documento (bitácora) en el que se detallan todos los pasos realizados desde el primer contacto con la comunidad, hasta que se finalizó la participación o se redactó este documento para los casos en los que aún se mantiene una participación activa con la comunidad. La bitácora abarca la comunicación con la comunidad, información sobre la aplicación, herramientas utilizadas, etc. La bitácora completa sobre la participación en el Caso #5 se describe en el Anexo H, Bitácora de Colaboración en Oportunidad OpenHatch.

4.4.5.3. Resultados

Es importante analizar previamente el sistema utilizado para buscar colaboraciones que permitió encontrar este caso. Se trata de la página openhatch.org, descrita en el Anexo H, Bitácora de Colaboración en Oportunidad OpenHatch. Dentro de las funcionalidades relacionadas con OSS que contiene esta página, está la posibilidad de realizar búsquedas de colaboraciones en proyectos OSS, basadas en el lenguaje de programación en el que se deberían desarrollar.

Realizando una búsqueda sobre el lenguaje PHP se encontró con una ‘oportunidad’, como se denomina en la página, de desarrollar una implementación en un proyecto OSS que se estaba creando. En la descripción de la aportación se incluía una dirección de correo a la que se escribió para ofrecer nuestra colaboración, como muestra la Figura 4.5.

> Hi I am a student of computer science concerned with the development of the functionality described in this task. I guarantee a good job, because I know very well the language used. Thanks for your time, I hope your answer. Carlos

Figura 4.5: Solicitud de Oportunidad de Implementación en el Caso #5

Al poco tiempo se recibió una contestación afirmativa y se iniciaron una serie de comunicaciones para la realización del desarrollo. Todo esto se encuentra descrito en el apartado referente a este caso del Capítulo 5.

4.4.5.4. Conclusiones del Caso

Resultados de la participación: Se corrobora la facilidad que existe para dar con una oportunidad de aportación. Cualquier usuario con conocimiento de programación puede

iniciar una colaboración en un breve espacio de tiempo. Otra característica a tener en cuenta es que existen otros medios para llegar a los proyectos aparte de sus páginas o comunidades existentes.

Observando la aplicación de gestión de oportunidades hay que destacar el elevado número de participantes y la gran cantidad de mejoras en las que se puede trabajar, del orden de miles.

Opinión personal: Sin lugar a dudas ésta es la forma más sencilla de encontrar proyectos en los que trabajar. El único problema es que al filtrar por lenguajes, la temática del proyecto no tiene por qué ser del agrado del desarrollador.

Al trabajar sobre un proyecto que aún no está desarrollado y del que solo se tiene la referencia de los *mails* que se leen en la lista de correos, surge la duda de cómo asegurarse que el código utilizado va a ser utilizado de forma correcta. Por este motivo se estudiarán más adelante las licencias existentes.

CAPÍTULO 5.

ANÁLISIS DE REQUISITOS, DISEÑO E IMPLEMENTACIÓN EN PROYECTOS OSS

En las actividades de análisis de requisitos, diseño e implementación se definen las necesidades que se van a satisfacer, cómo y qué se va a llevar a cabo y se produce también el desarrollo de los requisitos de software. En el trascurso de proyectos OSS se pueden desarrollar estas actividades como medio para resolver un error o con motivo de añadir mejoras y nuevas funcionalidades. Se trata por tanto, en muchos casos, de una tarea individual, ya que el desarrollo la mayor parte de las veces lo realiza un único usuario. Sin embargo, hay que prestar atención a ciertos detalles de estas actividades, tales como la educción de requisitos o la implementación, ya que pueden ayudar a una mejor evolución del proyecto.

Este capítulo pretende mostrar cómo se han desarrollado estas actividades mediante la participación en la implementación de algunas funcionalidades. También se van a mostrar las herramientas utilizadas y las comunicaciones realizadas con la comunidad. El análisis se desarrollará mediante los dos casos expuestos anteriormente, Caso #4 y Caso #5, en los que la comunidad OSS nos ha permitido participar.

5.1. Planteamiento del Problema

Tras estudiar de forma teórica y práctica las actividades de análisis de requisitos, diseño e implementación en un PDT, surge la necesidad de ampliar el conocimiento, únicamente teórico, del funcionamiento de un proyecto OSS en esas actividades. En lo referente al OSS se desconoce el funcionamiento de estas actividades en la práctica, por ejemplo si se utiliza algún sistema de control de versiones, si existe la posibilidad de que trabajen varios usuarios al mismo tiempo, si se entregan documentos donde se define el diseño, quién y cómo se valida que el trabajo es correcto, etc. Para resolver estas incógnitas se procede a exponer las participaciones realizadas en proyectos OSS.

Como usuario, ya miembro de varias comunidades OSS y habiendo participado en algunas de sus actividades, se ha conseguido la responsabilidad de corregir un error existente y de desarrollar un conjunto de funcionalidades para proyectos OSS. Así se consigue introducirse dentro de las actividades de análisis de requisitos, diseño e

implementación, lo que será de vital importancia para el análisis comparativo de estas actividades en relación con las de un PDT.

5.2. Objetivos de Investigación

El objetivo de esta investigación es el de conocer la manera en la que operan las comunidades OSS en las actividades de análisis de requisitos, diseño e implementación. Esto incluye conocer las tareas realizadas durante la implementación.

Con esto se pretende comprender de forma práctica como un usuario puede llevar a cabo una colaboración con una comunidad OSS desarrollando nuevas funcionalidades o corrigiendo errores, para una vez finalizado el análisis, poder comparar la forma de proceder en el proceso de desarrollo de un proyecto OSS frente a un PDT.

5.3. Diseño del Estudio de Casos sobre las Actividades de Análisis de Requisitos, Diseño e Implementación

5.3.1. Preguntas de Investigación

¿Cómo se proponen y especifican requisitos funcionales y no funcionales? ¿Se realizan documentos con requisitos, diseños,…? ¿Cómo se puede acceder al código fuente? ¿Se utiliza un sistema de control de versiones? ¿Cómo se asignan tareas a los usuarios para que éstos las implementen? ¿Quién valida la corrección del código implementado? ¿Existen algunos estándares de codificación que haya que seguir? ¿Es el usuario libre de implementar cualquier cosa? ¿Pueden dos usuarios trabajar en la misma tarea? ¿Pueden los usuarios seleccionar el lenguaje de programación? ¿Está correctamente estructurado y comentado el código de los proyectos?

5.3.2. Procedimiento de Análisis

En este apartado, para obtener una mejor descripción de los casos se va a proceder realizando una exposición detallada de todo el proceso, desde que se consiguió el desarrollo hasta que la comunidad lo dio por válido.

Previamente se situará el contexto del caso y al terminar, se analizarán los aspectos más relevantes y las conclusiones u opiniones personales a raíz de la participación.

5.3.3. Validación

El estudio de esta actividad está basado en las preguntas de investigación expuestas anteriormente. La respuesta a todas las preguntas servirá para considerar un caso como exitoso, ya que con todas las respuestas se podrá obtener una conclusión clara sobre las actividades de análisis de requisitos, diseño e implementación y por lo tanto la

información obtenida sería suficiente para poder comparar los dos modos de desarrollo de software.

Además, de forma personal y para la mejor consideración de este documento, se considera fundamental realizar una aportación a una comunidad que permita solucionar un error o añada alguna funcionalidad (Casos #4 y #5, respectivamente).

5.4. Descripción de los Casos

5.4.1. Identificación del Caso: Caso #4 GanttProject

5.4.1.1. Introducción y Contexto

GanttProject es una aplicación para la gestión de proyectos mediante la creación de diagramas de Gantt. Durante la actividad de exploración de conceptos se inició una comunicación con la comunidad mediante su *tracker*, lo que permitió la asignación de la corrección de un error. El error consistía en una seria de *pop-ups* que salían en las opciones del menú y que dificultaban la visión de otros elementos.

A continuación se exponen los pasos que se han seguido para desarrollar la funcionalidad comentada, para poder finalizar analizando las actividades que se estudian en este apartado, análisis de requisitos, diseño e implementación. La bitácora completa sobre la participación en el Caso #4 se describe en el Anexo G, Bitácora de GanttProject.

5.4.1.2. Resumen y Resultados

En primer lugar se procedió a buscar toda la información que permitiese montar satisfactoriamente el entorno de desarrollo. Todos los datos existentes en las diferentes páginas de la aplicación indicaban que el lenguaje utilizado principalmente para el desarrollo de esta aplicación era Java. Para su implementación se decidió utilizar el software Eclipse, ya que incorpora el entorno de desarrollo llamado Java Development Toolkit (JDT), que viene incorporado de forma básica y que permite una fácil gestión del código. El sistema operativo en el que se va a llevar el desarrollo es Windows, por lo que hay que prestar especial atención en que las herramientas utilizadas funcionen correctamente en este entorno.

El siguiente paso es obtener el código fuente, para ello existen dos opciones, descargarlo directamente de la página del proyecto o descargarlo desde un software de control de versiones. Observando los comentarios de errores ya resueltos, para poder obtener información de cómo llevar a cabo la tarea de la mejor forma posible, se concluyó que el mejor sistema era el de control de versiones, ya que permitía descargar y subir las modificaciones de forma más rápida y mejor documentada, lo que es de vital importancia para el correcto funcionamiento de la comunidad. El software utilizado es Mercurial, un sistema de control de versiones multiplataforma para desarrolladores de

software, que fue implementado originalmente para funcionar sobre Linux, pero que ha sido adaptado para Windows y otros sistemas. Las principales características de Mercurial son su gran rendimiento, escalabilidad que permite un desarrollo completamente distribuido, la gestión robusta de archivos tanto de texto como binario y otras capacidades avanzadas de ramificación e integración. Tras la instalación de esta herramienta se descargó el código de la aplicación y se dispuso a analizar los cambios a realizar.

Como se ha comentado anteriormente, la corrección del error consistía en eliminar los *pop-ups* de las opciones del menú. Estos *pop-ups* de texto aparecían en el menú principal y no tenían prácticamente ninguna utilidad. El error había sido reportado por otro usuario, por lo que se probó la aplicación para comprobar cuál era el problema real, ya que para poder solucionarlo de la mejor forma posible es necesario comprender bien en qué consiste el error.

Una vez analizado el error a reparar, se procedió a buscarlo en el código, para poder estudiar cómo llevar a cabo su corrección. El código se encontraba bien organizado, pero de forma inicial se decidió realizar una búsqueda en todos sus ficheros de una serie de palabras clave, como *pop-up, tooltip* o menú. Observando los resultados, se encontró el código que mostraba los *pop-ups*. Encontrado el código se procedió a realizar el cambio ya que se consideró una modificación sencilla.

Tras probar que se obtenía el resultado esperado se preguntó en la comunidad como subir las modificaciones realizadas. La respuesta exponía que en caso de tratarse de un código que afecta a uno o dos archivos, la mejor forma era mandarlo por email. Al ser éste el caso, se envió el fichero modificado a un administrador. Enseguida se recibió una contestación en la que se informaba que el cambio no había sido realizado de forma correcta, ya que borraba los *pop-ups* no solo de los menús sino de todos lados, lo que obviamente no era una opción. Se asumió el error y se procedió a analizar más detenidamente cuál era el cambio que se debía hacer, haciendo un uso más detallado de la aplicación para comprender el alcance real de los cambios realizados anteriormente. Una vez desarrollada la solución correcta y tras probarla exhaustivamente, se volvió a enviar para su aprobación.

En esta ocasión se dio por buena y se cerró por tanto la incidencia, obteniendo el agradecimiento del descubridor del error. Hay que destacar que el administrador además de comprobar que todo era correcto, nos comentó por correo que había modificado parcialmente los cambios, principalmente en estilo y forma.

Los cambios realizados y, por tanto, el error corregido aparecen en la versión 2.6.5.

5.4.1.3. Conclusiones

En primer lugar hay que citar que con este caso se ha conseguido el objetivo principal de colaborar activamente en una comunidad OSS. Aparte de este éxito, hay que analizar desde el comienzo el caso, ya que existen una serie de detalles que pueden ayudar a obtener una idea del funcionamiento de las actividades tratadas en este apartado dentro de un desarrollo OSS.

En primer lugar, para poder desarrollar hay que analizar las herramientas que se necesitan así como descargar el código de la aplicación. Una vez se tienen todo preparado para poder realizar la implementación, hay que analizar los requisitos. En este caso no se ha redactado ningún documento, simplemente se tiene la descripción del error creada por el usuario que lo reportó. Eso puede ser un problema, ya que, en muchos ejemplos que se analizaron, las descripciones eran muy ambiguas.

En este caso no fue necesario realizar un diseño, ya que las modificaciones a realizar no eran muy complejas y no existía ningún elemento, por ejemplo visual, que requiriese alguna actividad propia de esta actividad. Se podría decir que el diseño se resumió en decidir la estrategia para proceder a realizar los cambios, pero en ningún caso se dejó constancia de ello, fue simplemente una actividad a nivel personal.

A la hora de la implementación hay que destacar la falta de comentarios en el código. Como se ha analizado en el estado de la cuestión, la codificación es una forma de documentar los proyectos y es fundamental para la comprensión del código, sobre todo si lo deben interpretar otras personas. En este caso apenas existían comentarios, lo que dificultó la comprensión. Dado el tamaño del código realizado no se pudieron usar ningún tipo de patrón, simplemente se decidió como ejercicio personal comentar debidamente el código implementado, pero en ningún caso esto fue un requisito impuesto por la comunidad.

Para terminar, hay que destacar la ayuda recibida y el tiempo de respuesta. Se tuvieron una serie de problemas a la hora de enviar el parche y de comprender la estructura del código y la respuesta fue prácticamente inmediata. Otro aspecto importante es que en todo momento hubo un administrador revisando que todo era correcto, al menos en cuanto a funcionalidad. Se cree que quizás, ya que se revisan los códigos, se podría prestar más atención a la forma en la que están implementados, ya que sino las versiones futuras serán cada vez más difíciles de controlar, debido a las múltiples aportaciones.

5.4.2. Identificación del Caso: Caso #5 Colaboración en Proyecto OSS Social Network Manager

5.4.2.1. Introducción y Contexto

Como se ha explicado en el apartado de exploración de conceptos referente al Caso #5, la colaboración con este proyecto se obtuvo mediante la página openhatch, que permite buscar oportunidades de desarrollo en proyectos OSS por lenguaje. Dentro de las opciones existentes, se encontró esta colaboración, que se trataba de un proyecto OSS sobre la gestión y el estudio de las redes sociales en cuanto a relaciones entre usuarios y relevancia, cuyo desarrollo aún no estaba finalizado y para el que se necesitaba ayuda de otros desarrolladores. Se escribió a la comunidad ya que se conocía el lenguaje y la funcionalidad requerida parecía interesante. En concreto se trataba de unas funciones PHP que mediante llamadas a la API de Twitter permitiesen obtener la relación entre dos usuarios.

A continuación se exponen los pasos que se han seguido para desarrollar la funcionalidad comentada, para poder finalizar analizando las actividades que se estudian en este apartado, análisis de requisitos, diseño e implementación. La bitácora completa sobre la participación en el Caso #5 se describe en el Anexo H, Bitácora de Colaboración en Oportunidad OpenHatch.

5.4.2.2. Resumen y Resultados

Una vez la comunidad, mediante correos, estuvo conforme en que yo desarrollase la funcionalidad, se llevaron a cabo una serie de comunicaciones para especificar el desarrollo que se quería realizar. Para ello enviaron por correo una descripción de la funcionalidad deseada, de la que se podía obtener una lista de requisitos. Como resumen, se pedía recibir dos usuarios como parámetros y devolver una serie mínima de personas que haciendo retweets uniera a los dos usuarios indicados. También se debería devolver un nivel, que indique el número de retweets que conlleva esta comunicación.

Aparte de esta descripción, se recibió una serie de funciones PHP en un archivo comprimido que permitían usar la base de datos (BBDD) de la aplicación. Estas funciones permitían insertar usuarios en la BBDD, recuperar los seguidores, insertar los usuarios a los que se sigue, etc. Estas funciones venían definidas cada una en un archivo PHP distinto, pero no se tenía acceso a él, solo se tenían sus descripciones.

Por último se recibió por mail un manual de uso de la Api de Twitter, en el que se explicaba el funcionamiento de las diferentes llamadas que podían resultar útiles. Otra de las explicaciones que se daban era acerca del límite de peticiones, 150, que podía resultar un problema, pero que se estaba trabajando para conseguir eliminar. En cualquier caso yo llevaría a cabo el desarrollo con el límite, la solución sería posterior.

Un dato relevante era que la implementación sería llevada a cabo en Javascript, no en PHP, lenguaje que se especificaba en la descripción de la página, pero como también se conocía no supuso un problema.

La primera petición que se me hizo fue la de llevar a cabo un diseño de cómo se pretendía resolver, para poder evaluar el rendimiento que tendría. Tras pensar en la mejor forma posible se redactó un documento en el que se explicaba el diseño propuesto, indicando las funciones utilizadas y el flujo que se seguiría, Figura 5.1. y Figura 5.2. Para realizar las búsquedas de los usuarios entre los seguidores, se empleó un algoritmo de búsqueda en anchura, ya que se creyó la forma más efectiva y apropiada. En poco tiempo se recibió una contestación en la que nos felicitaban por la solución propuesta y se nos daban una serie de recomendaciones para desarrollar el código, organizarlo y probarlo.

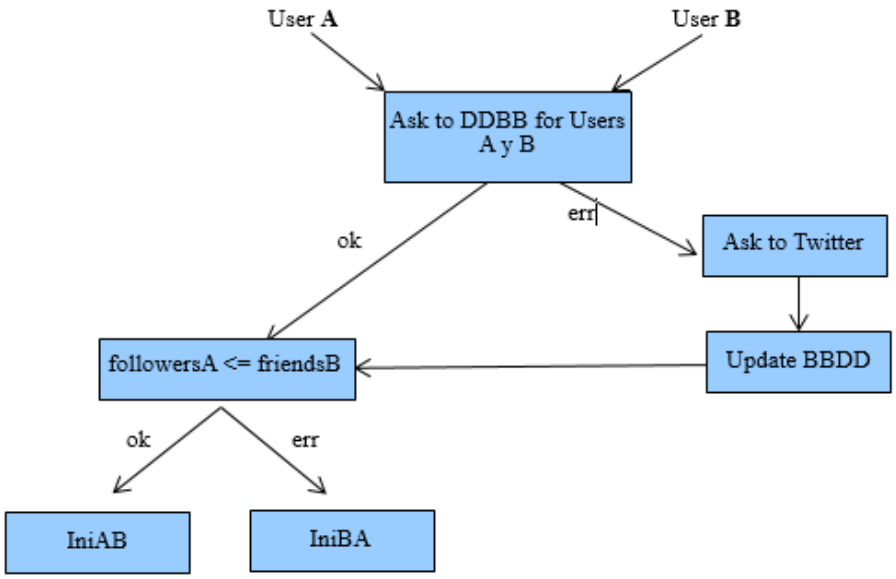


Figura 5.1: Esquema Inicio llamadas (Original)

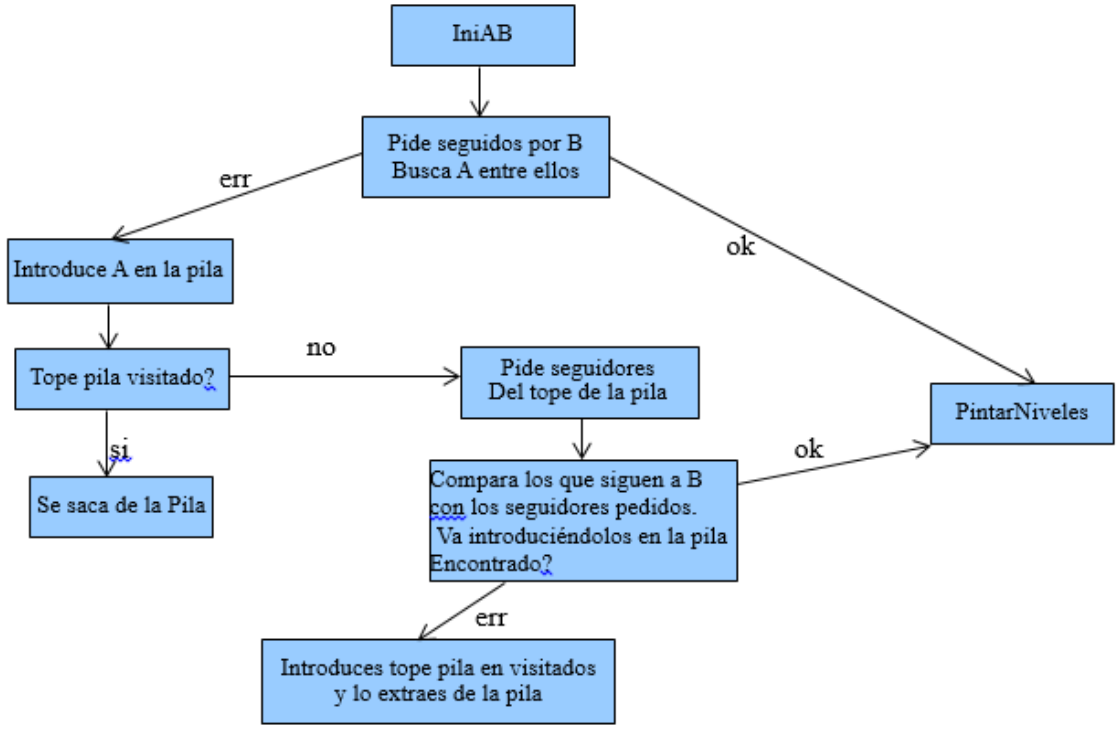


Figura 5.2: Esquema Función IniAB (Traducido)

A la hora de la implementación, surgió una duda acerca de cómo asegurarse que este código sería empleado de la forma deseada, como OSS. Para ello se estudiaron las licencias existentes y se decidió utilizar la General Public License GNU. Por supuesto se escribió a la comunidad para contar que se iba a aplicar esta licencia, a lo que contestaron diciendo que por supuesto no existía ningún problema. Ya que ese era el objetivo de este proyecto OSS.

Una vez desarrollado el código, se probó teniendo en cuenta las recomendaciones recibidas de la comunidad. Se creó un pequeño HTML que, obviando las consultas a la BBDD y basándose solo en las de Twitter, mostraba los resultados. Tras varias pruebas y ajustes se envió a la comunidad. Al cabo de unas semanas nos contestaron explicando que funcionaba correctamente y que estaban muy agradecidos. De hecho preguntaron por mi disponibilidad para continuar con la participación en un futuro, a lo que obviamente se contestó que sí.

En este punto se redactó este documento por lo que las colaboraciones que se realicen a partir de ahora serán trabajo futuro de interés personal y por tanto no aparecen en este caso de estudio.

5.4.2.3. Conclusiones

En primer lugar hay que citar que con este caso se ha conseguido realizar una aportación previa a la publicación del proyecto OSS, algo no muy usual y que sin lugar

a dudas nos permite analizar de forma más eficiente los métodos de trabajo de las comunidades OSS.

Nada más comenzar a trabajar con la comunidad en este proyecto se recibió un correo con los requisitos de la funcionalidad a analizar. Se trataba de una simple descripción, de cómo debía responder la aplicación, no seguía ningún tipo de estándar ni contenía un listado con los requisitos separados según el tipo, era un texto en el que se especificaba de forma detalla el módulo a implementar. Esa información fue suficiente para comprender qué se me pedía.

En cuanto al archivo comprimido de las funciones PHP con las que contaba, me sorprendió lo bien documentadas y organizadas que estaban, algo que sin duda facilitaba su uso para usuarios que no conociesen su código.

Una vez que se empezó con el desarrollo, me resultó extraño el error que hubo con el lenguaje a utilizar. En la página, cuya principal cualidad era la búsqueda por lenguaje, se especificaba que la implementación sería en PHP, pero en realidad iba a ser Javascript. No hubo ningún problema pero es algo que se debe cuidar ya que puede resultar una pérdida de tiempo para la comunidad e incluso puede llevar a confusiones que produzcan códigos inservibles. El desarrollo se llevó a cabo sin problemas, se tardaron unas dos semanas de forma bastante activa y se probó creando una página web. Hay que destacar como la comunidad pedía de forma insistente que el código se probase, suponemos para no perder ellos tiempo montándolo y probándolo varias veces. Otro aspecto interesante son las recomendaciones recibidas para la implantación del código, como funciones con un nombre que sirva de referencia, comentario en el código si se cree necesario, etc. No se pidió seguir ningún estándar, simplemente sentido común teniendo en cuenta que ese código debía ser legible para otros usuarios. En lo referente a la licencia, es importante analizar la contestación, ya que aseguraban que ese era el objetivo del proyecto, hacer OSS que pueda ser utilizado y mejorado por los propios usuarios de la aplicación.

Al terminar y enviar el código se recibió una contestación muy agradable agradeciendo el trabajo, algo que fue satisfactorio por mi parte.

Para terminar hay que evaluar esta colaboración como exitosa, ya que ha permitido conocer de forma práctica el desarrollo de un proyecto OSS y porque permitirá continuar con la colaboración.

CAPÍTULO 6.

PRUEBAS Y MANTENIMIENTO EN PROYECTOS OSS

Las actividades de pruebas y de mantenimiento son muy distintas a las otras analizadas anteriormente, debido a numerosos matices, pero principalmente porque ambas son transversales y engloban a todas las estudiadas en los capítulos previos. La actividad de exploración de conceptos así como las de análisis de requisitos, diseño e implementación forman parte de las actividades de mantenimiento, de hecho son consecuencia directa de las mismas. Las actividades de pruebas se realizan sobre los productos software y procesos involucrados en dichas actividades orientadas al desarrollo.

Por estos motivos no se van a analizar caso por caso, sino que se van estudiar a partir de todo el conjunto de la experiencia adquirida durante el desarrollo de los proyectos OSS en los que se ha participado.

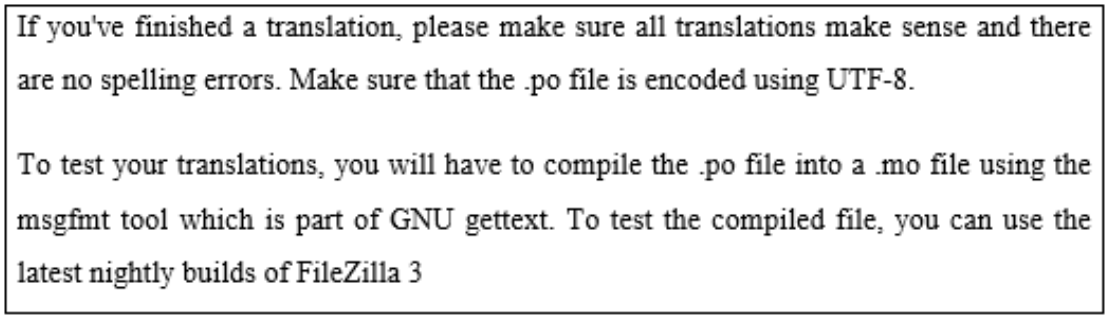
6.1. Pruebas en Proyectos OSS

Como se explicó en el estado de la cuestión, las pruebas son un conjunto de actividades de soporte dentro del desarrollo de software cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto. Tomando esta definición, dentro de los casos estudiados aparecen una serie de ocasiones en las que se han realizado actividades de pruebas sobre el proyecto OSS. El objetivo de este capítulo es describirlas y obtener las características que permitan comparar las diferencias entre los proyectos OSS y los PDT.

6.1.1. Pruebas de los Colaboradores sobre sus Aportaciones

Durante el desarrollo de este documento, se han realizado aportaciones a distintos proyectos OSS y en todos ellas las comunidades solicitaban que dichas aportaciones fuesen probadas debidamente.

En primer lugar nos encontramos con el Caso #2, FileZilla, en él se colaboró realizando una serie de traducciones de inglés a español. En el manual de envío de colaboraciones se explicaba de forma detallada, Figura 6.1, como se debía comprobar antes de enviar que todo era correcto.


If you've finished a translation, please make sure all translations make sense and there are no spelling errors. Make sure that the .po file is encoded using UTF-8.

To test your translations, you will have to compile the .po file into a .mo file using the msgfmt tool which is part of GNU gettext. To test the compiled file, you can use the latest nightly builds of FileZilla 3

Figura 6.1: Manual Carga de Traducciones en el Caso #2

Como se puede observar, se da vital importancia a realizar entregas de forma correcta, de hecho en la página de FileZilla se explicaba que de no cumplir una serie de características, el fichero sería rechazado.

Otro ejemplo es el del Caso #5, en el que la comunidad nos solicitó de forma directa que se probase el código e indicó unas pautas o consejos para hacerlo. Lo que proponían era crear un HTML básico en el que se incluyeran las llamadas a las funciones que se debían implementar, mostrando los resultados. Esto sirvió únicamente para uso personal, pero con ello se corroboró que la implementación carecía de fallos, al menos en lo funcional.

Con estos ejemplos es fácil ver la importancia que tiene comprobar que las colaboraciones que se realizan para una comunidad OSS sean correctas, ya que facilita de forma notable el trabajo de los administradores y permite avanzar de forma más rápida.

6.1.2. Pruebas de Validación del Trabajo de los Colaboradores

Los administradores de las comunidades son los encargados de asegurar que las colaboraciones que les envían los usuarios sean correctas. En algunos casos, si el código es sencillo de probar, son otros usuarios los que les ayudan validándolas por su cuenta.

Como se puede esperar, no se ha podido realizar esta tarea, ya que no se es administrador en ninguna comunidad, pero se puede usar el Caso #4 como ejemplo. En este caso se aportó un código para solucionar un error con unos *pop-ups*. La primera versión del código corregía este error pero producía una serie de fallos en otros elementos. El administrador que probó el código nos hizo saber al poco tiempo que no era una solución válida, lo que demuestra que no solo se dedicó a comprobar si se solucionaba el error, sino que analizó el código para ver si impactaba de forma negativa en otras funcionalidades.

Con este ejemplo se comprueba la implicación de los administradores con los proyectos OSS a los que pertenecen, ya que dedican su tiempo y esfuerzo a validar un gran número de aportaciones.

6.1.3. Pruebas sobre los Proyectos OSS

Todas las comunidades OSS en las que se ha trabajado cuentan con un sistema para que los usuarios que lo deseen puedan enviar los errores con los que se encuentran. En algunos casos se realiza directamente con listas de correo, otras cuentan con foros donde se pueden abrir temas relacionados con el error y la gran mayoría cuentan con sistemas de *Tickets*. Este sistema permite a los usuarios crear uno cuando vean un error e ir realizando un seguimiento.

Una vez que se reportan los errores, los administradores pueden aceptarlos o rechazarlos, ya sea porque no se consideran errores o porque el error ya ha sido reportado por otro colaborador y también pueden asignar un grado de severidad o prioridad a dichos errores. Una vez que se ha aceptado, los usuarios pueden dar su opinión, lo que sirve para evaluar el alcance del error o solicitar su asignación para corregirlo, como se hizo en el Caso #4, donde se pudo solucionar un error que había sido reportado por otro usuario.

6.2. Mantenimiento en Proyectos OSS

La actividad de mantenimiento de un software puede definirse como la modificación de un producto una vez que se ha entregado o se está desarrollando, ya sea tanto para corregir errores como para mejorar el rendimiento u otros atributos. A partir de esta definición es fácil deducir que las comunidades OSS están orientadas a realizar un mantenimiento inherente sobre los proyectos OSS.

Echando un vistazo a las colaboraciones realizadas, es fácil atribuir a cada una de ellas una actividad relacionada con el mantenimiento. En el Caso #2, por ejemplo, se realiza un cambio adaptativo, adaptando la aplicación a un idioma distinto al inicial. En este caso, al igual que en el Caso #4, también se encuentra un error que podía producir un comportamiento no deseado de la aplicación, siendo un cambio correctivo en la misma.

A lo largo de la búsqueda de casos en los que colaborar, se ha trabajado con decenas de comunidades OSS, observando como todos los proyectos que se encontraban en ellas tenían un gran número de versiones, todas con cambios aportados por los usuarios de las comunidades. Además, estas comunidades están preparadas precisamente para que sea sencillo a los usuarios reportar errores y proponer mejoras y también lo están para que los colaboradores puedan ir implantándolas de la forma más sencilla posible. Esto es la mejor forma de ver cómo, mediante el mantenimiento llevado a cabo por los

colaboradores de OSS, se ha conseguido evolucionar notablemente un enorme número de proyectos OSS.

Se puede concluir, por tanto, que una vez los desarrolladores de OSS suben un software a un repositorio y es accesible desde una comunidad, comienza un mantenimiento continuo basado en las funcionalidades que los usuarios creen que pueden hacer mejorar el software y en la corrección de los errores que contenga, haciendo que dicho software evolucione, en la mayoría de los casos, hacia donde los usuarios deseen.

CAPÍTULO 7.

COMPARATIVA ENTRE EL PDT Y El PDOSS

En este capítulo se van a analizar las diferentes actividades realizadas durante el desarrollo de proyectos OSS y tradicionales. Para ello se van a utilizar las experiencias recogidas durante la participación en un PDT, tanto en el desarrollo completo como en las pruebas sobre un *call center* de una Telco en la empresa Accenture y la participación en los diferentes proyectos OSS que se han descrito a lo largo de los capítulos anteriores, como por ejemplo VCL Media Player, GanttProject o FileZilla.

Esta comparativa, por tanto, se basará únicamente en la opinión creada a partir de la colaboración en ambos modelos, y servirá para contrastar estas vivencias con la teoría existente sobre el OSS, descrita en el Capítulo 2. La comparativa se realiza actividad por actividad para dejar ver de forma más clara las diferencias existentes.

En la actividad de **exploración de conceptos** existe un gran número de diferencias a tener en cuenta. En primer lugar, analizando un PDT frente a una colaboración de un usuario en una comunidad OSS encontramos una diferencia en el planteamiento. En el OSS el usuario de la aplicación propone un cambio en la misma para mejorar su funcionamiento, mientras que en el PDT es normalmente un cliente el que solicita el cambio o desarrollo. Esto que puede parecer lo mismo no lo es, ya que en OSS es el propio usuario el que se involucra en la mejora, pero en un PDT la responsabilidad está en manos de los responsables del software, que deben respetar un acuerdo con el cliente. Debido a esta responsabilidad adquirida, los PDT cuentan con modelos más rígidos, ya que se exigen documentos en los que se define lo que se va a desarrollar, cómo se va a desarrollar, si es viable técnicamente, etc. No se puede olvidar la parte económica, que aporta aún más formalidad y obliga a crear documentos de compromiso con el cliente, en los que se definen plazos de entrega. En los OSS no existe toda esta documentación, simplemente se llevan a cabo comunicaciones entre los miembros de la comunidad para decidir quién y cómo se va a llevar a cabo una propuesta de desarrollo, pero no se preocupa de la viabilidad técnica ya que se asume que la persona que desee desarrollarla está interesada y conoce cómo hacerlo. En lo referente a la viabilidad económica en los OSS no es importante, ya que el dinero en ningún caso es un elemento a tener en cuenta. Otro elemento que carece de importancia es el tiempo, la evolución o desarrollo de un cambio, por ejemplo, no tiene fecha de entrega, se finalizará cuando un usuario cree dicho cambio. Por último hay que evaluar los motivos por los que se rechaza un

proyecto, en el caso de los PDT pueden ser por motivos económicos, por no poder hacer frente tecnológicamente a la funcionalidad, etc. En un proyecto OSS las aportaciones se rechazan solo en caso de que no se crean oportunas o porque ya han sido realizadas.

En las actividades de **análisis y especificación de requisitos, diseño e implementación** vuelven a aparecer diferencias notables entre ambos modos de desarrollo.

El **análisis y especificación de requisitos** es una actividad fundamental en el PDT, ya que en ella se define completamente la funcionalidad de la aplicación, redactándose el documento de especificación de requisitos que servirá para validar el software. En proyectos OSS no existen tales documentos pues se supone que la propia aportación especifica de forma inequívoca lo que se pretende. En caso de no ser así se producirá un malentendido, que se corregirá mediante las comunicaciones entre usuarios. En el Caso #5, se realizó una especificación de requisitos, ya que los administradores de la comunidad dieron una descripción de la funcionalidad que se debía implementar, pero en ningún caso se utilizó ningún estándar de documentación o modelo para redactar dicha descripción.

La actividad de **diseño** en los PDT involucra también documentos con estándares y los diagramas, como modelos estructurados u orientados a objetos. El motivo es que normalmente colaboran varias personas, que no tienen por qué estar en el mismo lugar, por lo que el adecuado entendimiento del software es fundamental. Otro de los motivos es que la gran mayoría de las veces, los usuarios finales han de validar el diseño, por lo que todos estos elementos son de vital importancia. En los proyectos OSS el diseño es una actividad propia del usuario que decide implementar una aportación o corregir un error, por lo que no tiene la necesidad de realizar esquemas o modelos. Si lo desea, puede plantear la solución o preguntar por otras soluciones existentes a los miembros de la comunidad, que colaborarán con sus ideas, pero la decisión de diseño la tomará él.

La **implementación** en PDT tiene características similares a las anteriores actividades. A la implementación se le exigen una serie de cualidades dependiendo del proyecto, como que sea sencilla de mantener, que el código esté correctamente comentado y por supuesto que existan documentos técnicos donde se especifiquen las características del código. Esto permite y es consecuencia de la numerosa participación de programadores en un mismo PDT, así como para favorecer tareas futuras sobre esa misma implementación. En los proyectos OSS nos encontramos con un planteamiento totalmente distinto ya que no se realizan ninguno de esos documentos y el código carece de comentarios más allá de la codificación inicial y de algún usuario que los incorpore. Eso sumado a las numerosas aportaciones hace que poco a poco el código sufra modificaciones y sea más complicado de comprender.

Durante el desarrollo y una vez se ha desarrollado el software, es el momento de realizar las actividades de **pruebas**. En los PDT las pruebas se definen mediante documentos

como los planes de pruebas, que servirán para que el usuario valide el funcionamiento. En los proyectos OSS son los usuarios finales los que prueban el software y reportan los errores encontrados durante el uso. Lógicamente los colaboradores y los administradores de las comunidades realizan pruebas sobre las implementaciones antes de publicarlas, pero éstas no están definidas en ningún documento. Las pruebas realizadas por los usuarios no garantizan que la aplicación no contenga errores ya que solo se solucionarán aquellos que afecten directamente a los usuarios.

Por último está la actividad de **mantenimiento**. En los PDT, la existencia de un mantenimiento depende de varios factores. Si se trata de un software de tipo comercial, se realizan actualizaciones que solucionan los problemas que se van encontrando y donde se añaden algunas nuevas funcionalidades, ya sea para mejorar el servicio como para adaptarlo a nuevas necesidades. La otra opción es que sea un software realizado para un cliente y éste firme un contrato de mantenimiento. En este caso las actividades de mantenimiento que se realicen dependen de lo estipulado. En los proyectos OSS, una vez publicado el software y añadido a una comunidad, comienza el mantenimiento. Como se ha comprobado en la práctica, los repositorios contienen un gran número de herramientas para que los usuarios de las aplicaciones reporten errores o aporten ideas para mejorar el software. Además se incluyen herramientas para realizar cambios de tipo adaptativo, como pueden ser las traducciones o manuales de uso.

En resumen se puede decir que los modelos para los PDT son más rígidos y formalizados y contienen una gran número de especificaciones y de documentación, ya que están pensados para garantizar tiempos de entrega, realizar un uso efectivo de recursos y facilitar la colaboración con el usuario y el trabajo entre equipos de personas, que pueden no estar en el mismo sitio. Los proyectos OSS y sus métodos de trabajo están pensados para mejorar en función de las necesidades de los usuarios. Por este motivo utilizan herramientas que facilitan la comunicación entre los responsables y los usuarios finales, ya que en OSS el software avanza dependiendo del compromiso de los mismos.

A continuación, en la Tabla 7.1 se van a contrastar las características teóricas de la tabla realizada en el estudio conceptual del Capítulo 2 con las experiencias adquiridas tras colaborar en un PDT y en varios proyectos OSS. En fondo gris se consignan las experiencias personales obtenidas tras la participación en los procesos de desarrollo analizados.

Actividad	Características del PDT	Características del PDOSS	Experiencia Personal
Exploración de Conceptos	- Evaluación de la situación problemática existente - Definición de los objetivos preliminares a alcanzar - Estudio de la factibilidad técnica y económica - Busca un beneficio económico	- Busca satisfacer una necesidad - La viabilidad es decidida por los usuarios de la comunidad, que determinan qué es correcto para la comunidad - Altruista	En los proyectos OSS se busca mejorar el software mediante la aportación de los usuarios, pero en muchos casos las decisiones las toman los administradores. No se puede considerar altruista ya que se busca un beneficio personal. En OSS a diferencia del PDT no existen fechas para la finalización de las actividades, más aún en OSS no están previstas.
Análisis y Especificación de Requisitos	- Educción de requisitos - Requisitos contractuales - Roles específicos y poco numerosos participan en las tareas de decisión - Existen actividades de validación de requisitos	- Los usuarios piden nuevas funcionalidades - Requisitos estipulados de forma escrita sobre la marcha - Gran cantidad de personas participan a la hora de decidir - No existen actividades de validación de requisitos	En los OSS participan muchas personas pero la decisión final también reside en unos pocos, que tienen mayor conocimiento del software. Es cierto que no se validan requisitos a partir de un documento, ya que se definen de forma implícita.
Diseño	- Se dedica tiempo a pensar en el diseño - El diseño tiene el mismo peso que la implementación - Sólo los desarrolladores participan en el diseño y bajo unas definidas asignaciones de trabajo	- No se piensa mucho en el diseño. Se mezcla con la actividad de implementación - El diseño tiene menos peso que la implementación - Tanto los desarrolladores como lo usuarios participan en el diseño	En el PDOSS el diseño se incorpora dentro de la implementación y depende de cada usuario, en el PDT es clave ya que se busca la aceptación y la facilidad de distribución del trabajo.
Implementación	- Equipos de desarrollo centralizados - Tareas asignadas a los equipos - Sólo los desarrolladores acceden al código	- Equipos de desarrollo distribuidos - Los usuarios eligen qué implementar - Todo el mundo puede acceder al código	En OSS el colaborador decide cómo realizar la implementación y el grado de implicación en hacerla más comprensible para los demás desarrolladores. En el PDT es vital una buena documentación.
Pruebas	- Uso de *service packs* para solucionar los errores - Planes de prueba previamente estipulados	- Los usuarios reportan los errores encontrados y en algunos casos también los solucionan	En PDT las pruebas están definidas y el cliente o usuario las valida. En OSS los usuarios van reportando los errores a medida que utilizan el software.

Tabla 7.1: Características y Experiencia Personal en PDT y PDOSS por Actividad

Actividad	Características del PDT	Características del PDOSS	Experiencia Personal
Mantenimiento	- En el PDT, la identificación de las mejoras se realiza sobre la base de diferentes documentos, resultado de otras actividades relacionadas con la planificación del proyecto - En el desarrollo tradicional, los usuarios solo reportan errores, no los corrigen	- Los miembros del equipo núcleo de desarrolladores son los que deciden si incorporan o no una nueva funcionalidad - Cualquier persona, en cualquier momento, puede sugerir o aportar mejoras - Los usuarios finales OSS que actúan como desarrolladores o como encargados de realizar el mantenimiento producen continuamente estas mejoras - Los reportes de problemas o solicitudes de mejoras pueden ser realizados por cualquier persona, incluidos los usuarios	El proceso de mantenimiento tradicional del software no tiene nada que ver con el OSS, ya que éste se puede decir que se trata de un mantenimiento continuo y completo. En PDT depende de lo estipulado y en muchos casos puede anularse dicho mantenimiento.

Tabla 7.1: Características y Experiencia Personal en PDT y PDOSS por Actividad (Continuación)

CAPÍTULO 8.

CONCLUSIONES Y TRABAJO FUTURO

Este capítulo detalla las conclusiones obtenidas en el presente trabajo de investigación y desarrollo donde se ha participado de forma activa en una serie de proyectos OSS así como en un PDT y el análisis de los resultados logrados en dichos estudios empíricos. Igualmente se exponen las oportunidades que quedan abiertas en la investigación para seguir avanzando en distintas líneas importantes e interesantes a fin de continuar ampliando los conocimientos que permitan beneficiar a la Ingeniería del Software mediante la comprensión y combinación de todos sus modelos de desarrollo.

8.1. Conclusiones

El trabajo presentado en este documento recoge un estudio de las diferencias entre el modelo tradicional de realización de proyectos software frente al modelo *open source*. El estudio se ha realizado partiendo de los conocimientos teóricos existentes, para buscar y comprobar después en la práctica las características más relevantes. Los objetivos planteados consistían en analizar desde dentro el funcionamiento de los proyectos OSS participando activamente en las diferentes actividades del proceso de desarrollo de software. Los resultados obtenidos de estas experiencias han permitido analizar las diferencias más significativas entre los dos modelos estudiados.

A continuación se exponen las conclusiones obtenidas tras la realización y análisis de este estudio.

Los desarrollos de OSS se inician para cubrir una necesidad mediante el desarrollo de un software y los usuarios colaboran para mejorar dicho software, buscando su beneficio, con lo que de forma indirecta benefician a la comunidad. Los PDT se inician también para cubrir una necesidad pero en este caso con un fin comercial, ya sea para satisfacer las necesidades de un cliente o para vender el software directamente a usuarios finales.

En los PDT un proyecto puede ser rechazado si los desarrolladores no tienen los medios o conocimientos necesarios para realizarlo o también si no se satisfacen las expectativas económicas. En los desarrollos OSS, los colaboradores proponen mejoras y son los administradores de la comunidad los que tienen la última palabra. Esto, si no se disponen de medios óptimos para que otros usuarios apoyen las propuestas, puede hacer

que la responsabilidad esté solo en unos pocos. Por este motivo, los sistemas de comunicación entre los miembros de una comunidad OSS son fundamentales. En relación con este aspecto se ha comprobado que el mejor sistema es el de tickets o posts y no las listas de correo que dificultan el seguimiento de los aportes.

Debido a los componentes económicos que rodean a los PDT, estos están obligados a cumplir con requisitos como fechas de entrega y necesitan estar muy bien definidos para que se pueda evaluar el resultado. También hay que tener en cuenta que los PDT pueden ser desarrollados por diferentes compañías al mismo tiempo, lo que obliga a realizar una buena y extensa documentación. La documentación empleada afecta a todas las actividades del desarrollo, desde la exploración de conceptos hasta el mantenimiento, con documentos de especificación de requisitos, diagramas de casos uso, diseño de clases, planes de pruebas, etc. Todos estos documentos suelen estar basados en estándares que garantizan su efectividad. Esto hace que los PDT sean desarrollos más rígidos ya que están obligados a ceñirse a dichos documentos, pero tienen la ventaja de que son más sencillos de comprender y analizar para los colaboradores implicados. En los OSS existe muy poca documentación y está orientada a que los colaboradores puedan instalar las herramientas necesarias para probar e implementar mejoras pero en ningún caso ayudan a entender el funcionamiento de las aplicaciones a nivel de desarrollo. Esto dificulta la actividad de los colaboradores ya que encontrar y entender la aplicación lleva mucho más tiempo que si existiesen documentos donde poder estudiarlo. Con el código sucede algo parecido ya que la estructura, modelo y comentarios dependen de cada usuario y por tanto, la colaboración de un gran número de ellos hace que el código pueda ser muy difícil de entender.

A la hora de mantener el software, el PDT vuelve a estar marcado por motivos externos, como el contrato que se estipule con el solicitante del software o con las necesidades comerciales de los desarrolladores. Una vez más se definen en documentos planes de prueba y de mantenimiento que determinan qué se va a probar y durante cuánto tiempo se va a mantener. En el caso de las pruebas existe una ventaja, ya que las realizan los mismos desarrolladores y por tanto tienen un mayor conocimiento del alcance de la corrección de los errores, el problema es que solo se prueba lo estipulado. Lo mismo ocurre con el mantenimiento, que al estar definido previamente hace que los cambios o mejoras que no estén contemplados puedan no realizarse. En el desarrollo de OSS son los usuarios y colaboradores los que prueban la aplicación y los que tienen la responsabilidad de reportarlos y corregirlos. El mantenimiento en OSS es la actividad principal ya que todas las actividades relacionadas se pueden englobar en él. Los usuarios deciden cómo mejorar la aplicación, esto hace que el mantenimiento adaptativo sea mejor en el OSS, por ejemplo para un PDT puede no ser necesario tener cientos de traducciones, ya que implica un coste, mientras que para un PDOSS simplemente con que un usuario desee una traducción en su idioma y la realice el software habrá mejorado. Una posible ventaja del PDT es que al utilizar una serie de fechas de entrega

y objetivos, se asegura una evolución del software mientras que en OSS la evolución depende del compromiso de los colaboradores.

Por último hay que analizar unas diferencias más difíciles de comprobar como puede ser la motivación de las personas implicadas. En los PDT los desarrolladores por ejemplo implementan los módulos que se les indica y en el lenguaje que corresponda, mientras que en OSS son los propios desarrolladores los que deciden en qué colaborar y escogen el lenguaje y proyecto que más les interese, siendo en muchos casos una actividad de mejora de un programa que utilizan con normalidad.

Por tanto, las conclusiones extraídas de este trabajo facilitan la comprensión de los métodos de trabajo en los PDT y en los proyectos OSS, sus actividades y el papel desempeñado por las personas implicadas. Con ello se han definido las diferencias entre ambos modos de desarrollo así como sus puntos más fuertes y débiles.

8.2. Trabajo Futuro

La continuidad de la investigación recogida en el presente trabajo se dirige, principalmente, a las siguientes líneas futuras:

- Continuar con la colaboración del Caso #5, Proyecto OSS Social Network Manager, hasta su publicación en un repositorio y en una comunidad OSS, teniéndome como desarrollador. Utilizar esta condición para realizar un análisis del funcionamiento de un proyecto OSS desde el punto de vista de los administradores, analizando las herramientas para gestionar las peticiones, probar los cambios y estudiando los métodos de toma de decisión. Este análisis se podría completar con el análisis paralelo en un PDT, formando parte o utilizando los conocimientos de alguien que desempeñe un papel de gestión, ya que en el análisis práctico de los PDT no se ha podido colaborar en estas actividades.

- Realizar un modelo de desarrollo de software mixto formado por los procesos de desarrollo *open source* y tradicional. Actualmente hay muchas compañías que están incorporando el OSS a sus desarrollos y la creación de un modelo que incluya las ventajas de ambos y que elimine sus inconvenientes puede ser de gran ayuda para la evolución de los procesos de desarrollo de software.

REFERENCIAS

Acuña, S.T., Castro, J.W., Dieste, O., Juristo, N. (2012). A Systematic Mapping Study on the Open Source Software Development Process. Proceedings of the 16th International Conference on Evaluation & Assessment in Software Engineering (EASE'12), Ciudad Real, Spain, pp. 42-46.

Dinh-Trong, T., Bieman, J.M. (2005). The FreeBSD Project: A Replication Case Study of Open Source Development. IEEE Transactions on Software Engineering, 31: pp. 481-494.

Hars, A., Ou, S. (2002). Working for Free? Motivations for Participating in Open-Source Software Projects. International Journal of Electronic Commerce, 6(3): pp. 25-39.

IEEE Std. 830:1998. (1998). IEEE Recommended Practice for Software Requirements Specifications.

IEEE Std. 982.1:1988. (1998). IEEE Standard Dictionary of Measures to Produce Reliable Software.

IEEE Std. 1219:1998. (1998). IEEE Standard for Software Maintenance.

IEEE Std. 1074:2006. (2006). IEEE Standard for Developing Software Life Cycle Processes.

ISO 8402:1995. (1995). Quality Management and Quality Assurance.

ISO 9001:2008. (2008). ISO Standard of Quality Management System.

ISO 9002:1994. (1994). ISO Model of Quality Assurance in Production, Installation and Servicing.

ISO 9003:1994. (1994). ISO Standard of Quality in Final Inspection and Test.

ISO 9004:2009. (2009). ISO Standard of Quality in Management.

ISO/IEC 12207:1995. (1995). Information Technology. Software Life Cycle Processes.

ISO/IEC/IEEE 42010:2011 (2011). Systems and Software Engineering. Architecture Description.

UNE 66-907-91. (1991). UNE Standar to Establishment of a Quality Manual.

Johnson, K. (2001). A Descriptive Process Model for Open-Source Software Development. Master. Thesis in Computer Science. Department of Computer Science. University of Calgary, pp. 131-144.

Mockus, A., Fielding, R.T., Herbsleb, J. (2002). Two Case Studies of Open Source Software Development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology, 11(3): pp. 309-346.

O'Mahony, S. (2003). Guarding the Commons: How Community Managed Software Projects Protect their Work. Research Policy, 32(7): pp. 1179-1198.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M. (2008). Systematic Mapping Studies in Software Engineering. Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, University of Bari, Italy, pp. 71-80.

Potdar, V., Chang, E. (2004). Open Source and Closed Source Software Development Methodologies. Proceedings of the 26th International Conference on Software Engineering, 19(6): pp. 105-109.

Reis, C.R., Mattos Fortes, R.P. (2002). An Overview of the Software Engineering Process and Tools in Mozilla Project. Proceedings of the Workshop on OSS Development, Newcastle Upon Tyne, UK, pp. 162-182.

Rooney, P. (2005). IBM Builds Dedicated Sales Channel for Red Hat, Novell Linux. [Online] Available: http://www.crn.com/software/ 175002626 [Accesed in June 2014].

Runesson, P., Höst, M. (2009). Guidelines for Conducting and Reporting Case Study Research in Software Engineering. Empirical Software Engineering, 14(2): pp. 131-164.

Satzinger, J.W., Jackson, R.B., Burd, D.B. (2000). System Analysis and Design in a Changing World. Thomson Learning.

Scacchi, W. (2001). Is Open Source Software Development Faster, Better and Cheaper than Software Engineering? Proceedings of the 23rd International Conference on Software Engineering, Toronto, Ontario, Canada, pp. 119-124.

Scacchi, W. (2004). Free and Open Source Development Practices in the Game Community. IEEE Software, 21(1): pp. 59-66.

Scacchi, W. (2005). Socio-Technical Interaction Networks in Free/Open Source Software Development Processes. In: Acuña, S.T., Juristo, N. (Eds.) Software Process Modeling, Springer, New York, pp. 1-27 .

Schweik, C.M., Semenov, A. (2003). The Institutional Design of Open Source Programming: Implications for Addressing Complex Public Policy and Management Problems. Journal First Monday, 8(1), Chicago. [Online]. Available: http://firstmonday.org/issues/issue8_1/schweik/index.html [Accesed in June 2014].

Senyard, A., Michlmayr, M. (2004). How to Have a Successful Free Software Project. Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC 2004), IEEE Computer Society, pp. 84-91.

Vixie, P. (1999). Software Engineering. In: de DiBona, C., Ockman, S., Stone, M. (Eds.) Open Sources: Voices from the Open Source Revolution, O'Reilly Press, pp. 131-144.

Weber, S. (2000). The Political Economy of Open Source Software. BRIE Working Paper, Nº 140, [Online]. Available: http://e-conomy.berkeley.edu/publications/wp/wp140.pdf [Accessed in June 2014].

ANEXOS

Anexo A. Estándar 830-1998 para una Buena Documentación de Requisitos

A.1. Descripción del Estándar 830-1998

En este estándar se describen recomendaciones de la IEEE para la realización del proceso de creación de la especificación de requisitos de software [IEEE Std. 830:1998].

Este documento realiza en primer lugar una serie de recomendaciones para la consecución de una buena especificación de requisitos, además de describir la estructura lógica de la especificación de requisitos del software.

A.2. Recomendaciones Destacadas

Las recomendaciones que se deben considerar son las siguientes:

- **Naturaleza**: La especificación de requisitos de software es una especificación para un producto software en particular, programa, o conjunto de programas que realizan ciertas funciones en un entorno específico. Son cuestiones básicas que deben ser tratadas;
    - Funcionalidad.
    - Interfaces.
    - Rendimiento.
    - Atributos: corrección, mantenibilidad, seguridad, etc.
    - Restricciones de diseño impuestas sobre la implementación.
- **Entorno**: La especificación de requisitos de software debe:
    - Definir correctamente todos los requisitos software. Un requisito software puede existir por la naturaleza de la tarea a resolver o por una característica especial del proyecto.
    - No debe describir ningún diseño o detalle de implementación.
    - No debe imponer ninguna limitación adicional al software.

- **Características**:

    - Correcta.
    - No ambigua.
    - Completa.
    - Consistente.
    - Clasificados los requisitos por importancia y/o estabilidad.
    - Verificable.
    - Modificable.
    - Rastreable.

- **Evolución**: La especificación de requisitos puede evolucionar según progresa el proceso de desarrollo del software.

- **Inclusión de diseño:** La especificación de requisitos de software no debe incluir generalmente cuestiones de diseño como:

    - Partición del software en módulos.
    - Asignación de funciones a módulos.
    - Descripción de flujos de información o control entre módulos.
    - Elección de estructuras de datos

A.3. Esquema del Documento de Especificación de Requisitos

La Figura A.1 muestra el esquema que debe seguir un documento de especificación de requisitos según el estándar que se está estudiando [IEEE Std. 830:1998].

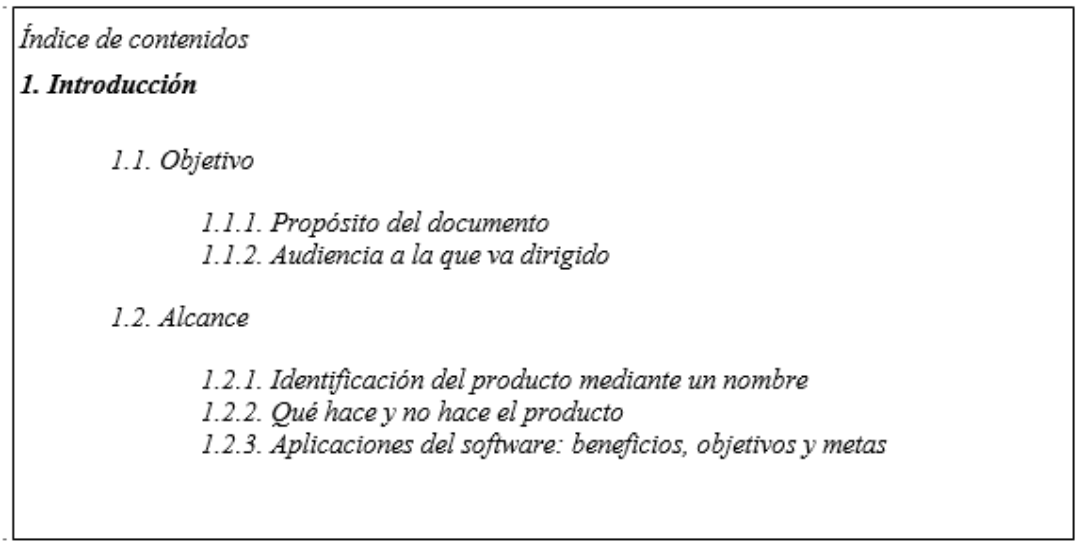


Figura A.1: Esquema Documento de Especificación de Requisitos

1.3. Definiciones, acrónimos y abreviaturas
1.4. Referencias
1.5. Visión general

1.5.1. Descripción del contenido del resto del documento
1.5.2 Organización del documento

2. Descripción general

2.1. Perspectiva del producto

2.1.1. Indicar si es un producto independiente o parte de un sistema mayor
2.1.2. Interfaces de sistema
2.1.3. Interfaces de usuario

2.1.3.1. Características lógicas del interfaz
2.1.3.2. Cuestiones de optimización del interfaz de usuario

2.1.4. Interfaces hardware
2.1.5. Interfaces software

2.1.5.1. Descripción del producto software utilizado
2.1.5.2. Propósito del interfaz
2.1.5.3. Definición del interfaz: contenido y formato

2.1.6. Interfaces de comunicaciones
2.1.7. Limitaciones de memoria
2.1.8. Operaciones

2.1.8.1. Modos de operación de los distintos grupos de usuarios
2.1.8.2. Periodos de operaciones interactivas y automáticas
2.1.8.3. Funciones respaldo del procesamiento de datos
2.1.8.4. Operaciones de backup y recuperación

2.1.9. Requerimientos para adaptarse a la ubicación

2.1.9.1. Indicar cualquier dato o secuencia de inicialización específico de cualquier lugar, modo de operación.
2.1.9.2. Características que deben ser modificadas para una instalación en particular.

2.2. Funciones del producto

Esta subsección debe proporcionar en resumen de las funciones principales que el software debe llevar a cabo. Las funciones deben estar organizadas de manera que el cliente o cualquier otro lo entienda perfectamente. Para ello se pueden utilizar métodos textuales o gráficos.

2.3. Características de usuario

Indica el tipo de usuario al que se dirige la aplicación: nivel de conocimientos, experiencia, etc.

Figura A.1: Esquema Documento de Especificación de Requisitos (Continuación)

2.4. Restricciones

Se debe indicar aquí cualquier tipo de limitación: hardware, seguridad, protocolos de comunicación, etc.

2.5. Suposiciones y dependencias

En este apartado aparecerá cualquier factor que afecte a los requerimientos. No se tienen en cuenta restricciones de diseño, por ejemplo, asumir que un determinado sistema operativo estará disponible.

2.6. Requisitos para futuras versiones del sistema

3. Requisitos específicos

Esta sección de la especificación de requisitos software contiene todos los requisitos hasta un nivel de detalle suficiente para permitir a los diseñadores diseñar un sistema que satisfaga dichos requisitos, y que permita diseñar las pruebas que ratifiquen que el sistema cumple con los requirimientos.
El estándar propone una serie de plantillas según el tipo de sistema con el que nos enfrentemos. En este caso se ha elegido el organizado según la jerarquía funcional que permite la inclusión de diagramas de flujos de datos y diccionarios de datos. La elección se realiza por eliminación, ya que como indica el documento, en el caso de que el proyecto no se ajuste a ningún otro esquema éste es el más adecuado.

3.1. Requisitos de interfaz externo

3.1.1. Intefaces de usuario
3.1.2. Interfaces hardware
3.1.3. Interfaces software
3.1.4. Interfaces de comunicaciones

3.2. Requisitos funcionales

3.2.1. Flujos de información

3.2.1.1. Diagrama de flujo de datos 1
3.2.1.2. Diagrama de flujo de datos .
. ..
3.2.1.n. Diagrama de flujo de datos n

3.2.2. Descripción de procesos

3.2.2.1. Proceso 1

3.2.2.1.1. Entidades de entrada de datos
3.2.2.1.2. Algoritmo o fórmula del proceso
3.2.2.1.3. Entidades de datos afectadas

Figura A.1: Esquema Documento de Especificación de Requisitos (Continuación)

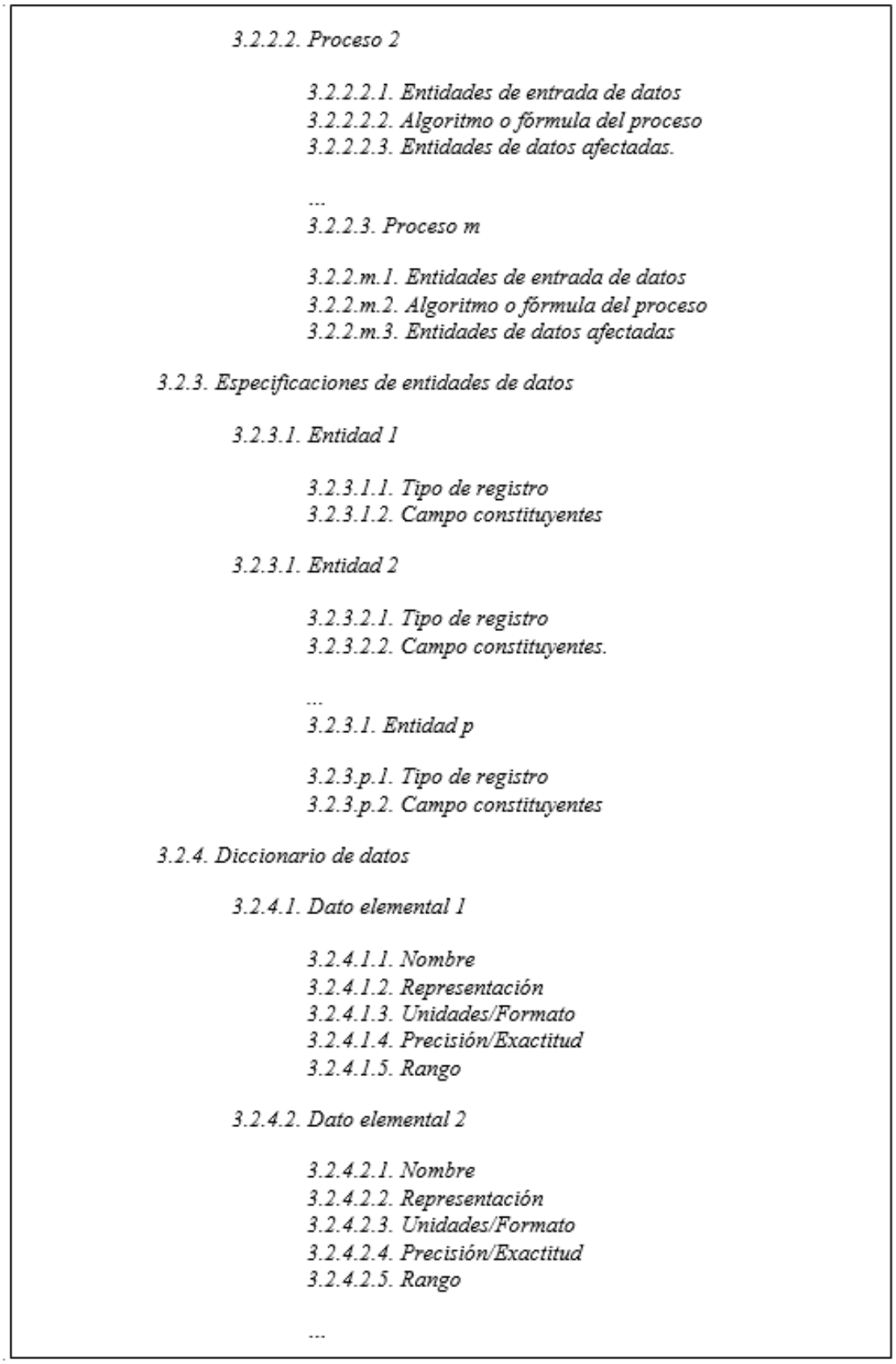


Figura A.1: Esquema Documento de Especificación de Requisitos (Continuación)

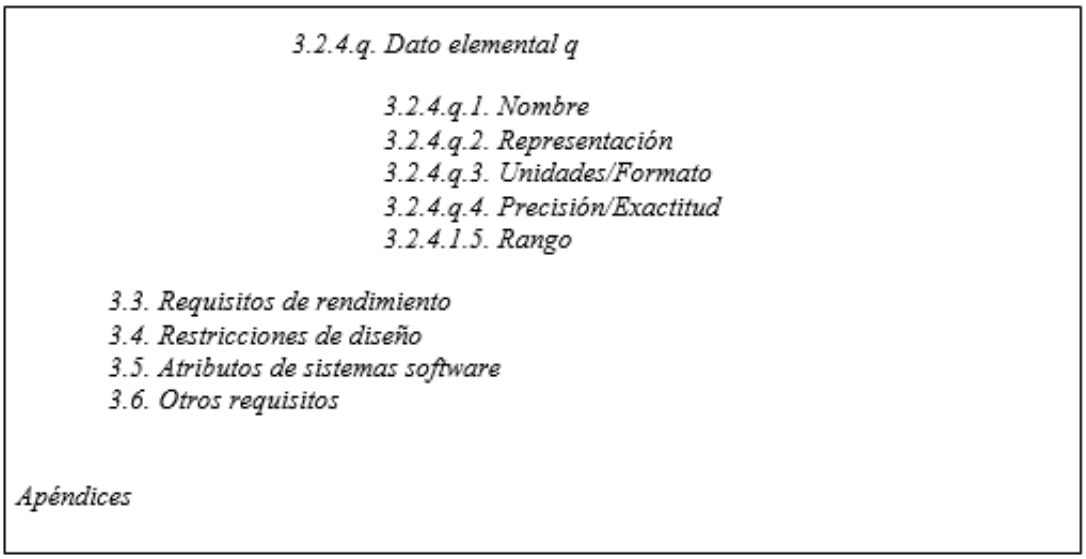


Figura A.1: Esquema Documento de Especificación de Requisitos (Continuación)

Anexo B. Estándares de Calidad ISO 9000

B.1. Descripción de los Estándares de Calidad ISO 9000

La denominada *International Organization for Standardization* (ISO), ha realizado un conjunto de estándares ISO 9000, que aseguran la calidad. Se pueden dividir en dos grupos de normas:

- **Normas para el aseguramiento externo de la calidad**, donde se puede encontrar: el [ISO 9001:2008] para el aseguramiento de la calidad en organizaciones cuyo proceso abarca desde el diseño hasta el servicio posventa; [ISO 9002:1994], que garantiza la calidad en organizaciones cuya actividad se basa en las fases de producción y de instalación; y el [ISO 9003:1994], que se centra en organizaciones cuya actividad consiste en inspecciones y ensayos finales.

- **Normas para la gestión interna de la calidad**, como el [ISO 9004:2009], que se basa en la norma [UNE 66-907-91:1991], que es una guía para manuales de calidad inspirada en la normativa nuclear.

B.2. Esquema Estándares de Calidad

El esquema de la Figura B.1 facilita la comprensión de la organización de dichos estándares de calidad.

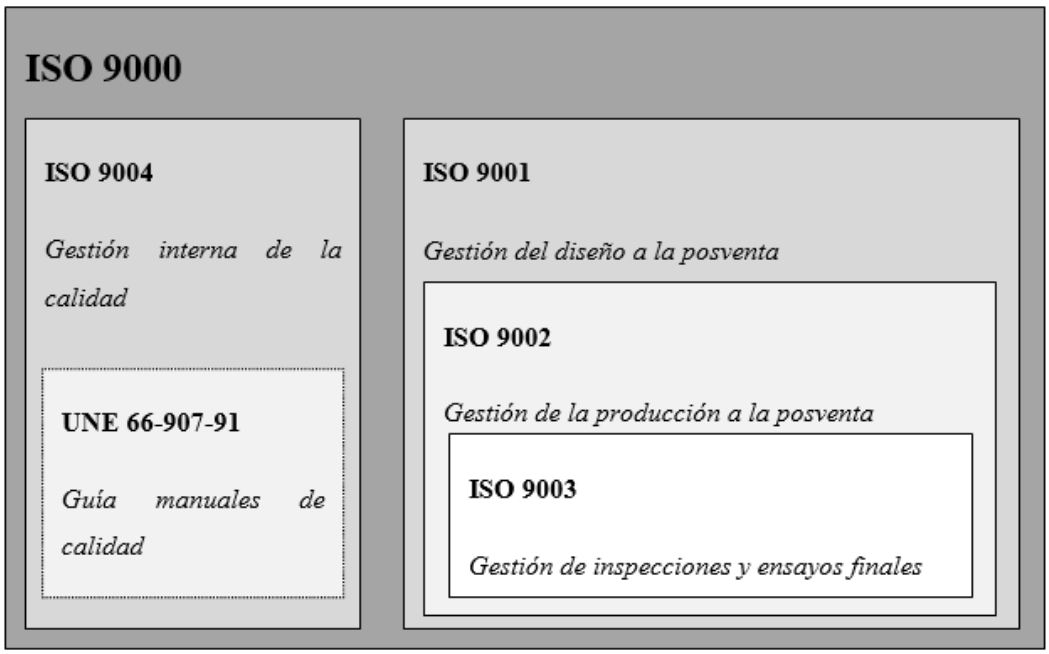


Figura B.1: Esquema Organización de Estándares de Calidad ISO 9000

Anexo C. PDT Real: Accenture Madrid

C.1. Descripción del Proyecto

La empresa consultora Accenture realiza el desarrollo de un sistema de *Call Center* para una conocida compañía de telecomunicaciones. El sistema sirve para realizar altas de usuarios y para la gestión de ofertas y contrataciones. Dentro del proyecto, se colabora desde el departamento de pruebas, lo que permite tener acceso a toda la documentación del desarrollo del proyecto. Las pruebas realizadas así como los documentos expuestos son relativos a una nueva funcionalidad que se desea añadir al sistema general, en concreto se trata de una nueva promoción para los contratos de tecnología ADSL.

En los siguientes apartados se detallan los documentos realizados en las diferentes actividades así como los correos electrónicos que se consideran más relevantes para la correcta comprensión de las actividades realizadas.

Debido a políticas de privacidad no se muestran los documentos completos, sino los apartados más útiles para el estudio realizado en este proyecto.

C.2. Documento Actividad de Exploración de Conceptos

La primera actividad que se lleva a cabo es la de exploración de conceptos. En ella, Accenture se reúne con el cliente para detallar la viabilidad de la necesidad expuesta por el usuario, pactar unas fechas de entrega y métodos de desarrollo y presupuestar el coste total. Una vez se llega a un acuerdo se redacta un documento, Figura C.1, a modo de contrato y que servirá para la evaluación del proyecto.

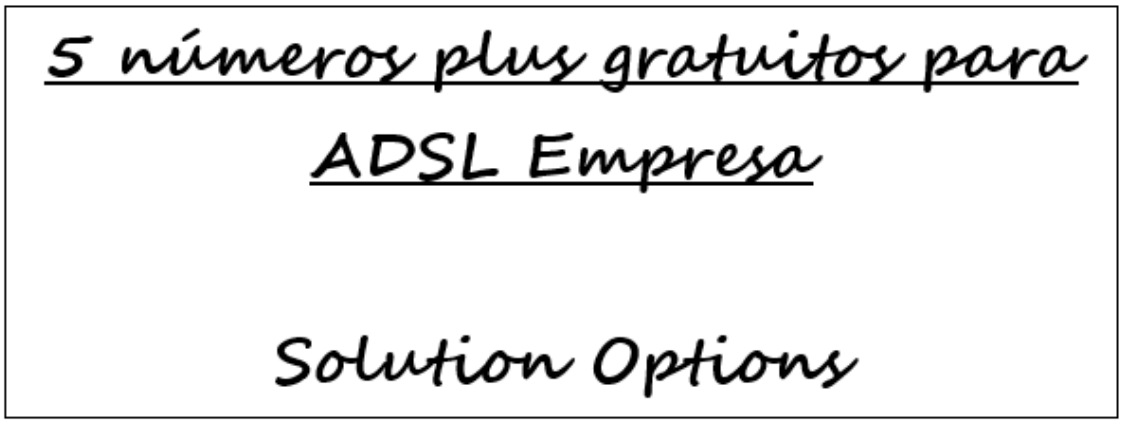
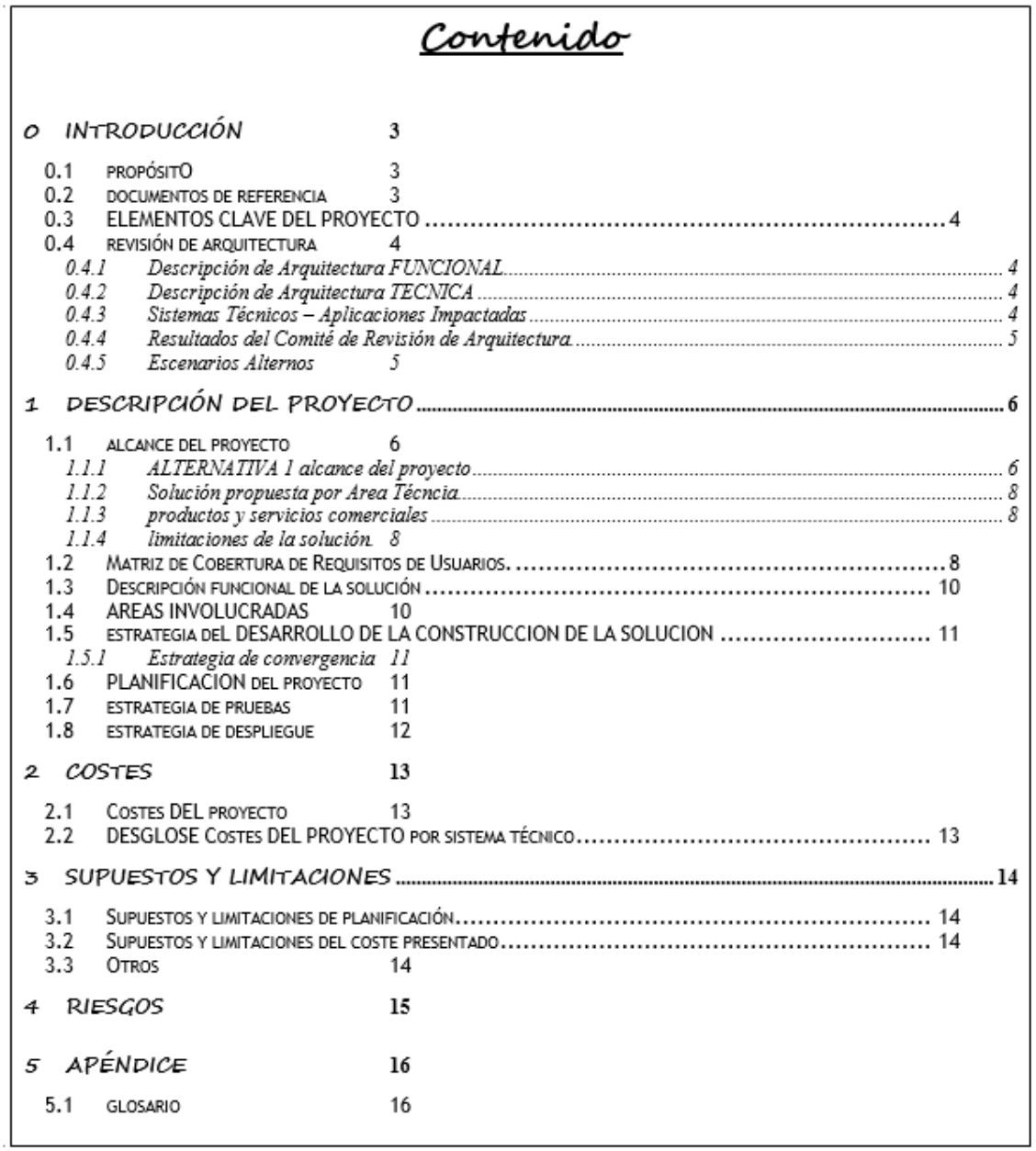


Figura C.1: Título Documento Actividad de Exploración de Conceptos

Observando la estructura del documento, Figura C.2, se puede comprobar los aspectos del proyecto que se describen. Inicialmente se expone el propósito, Figura 3.2, y los elementos clave del proyecto.

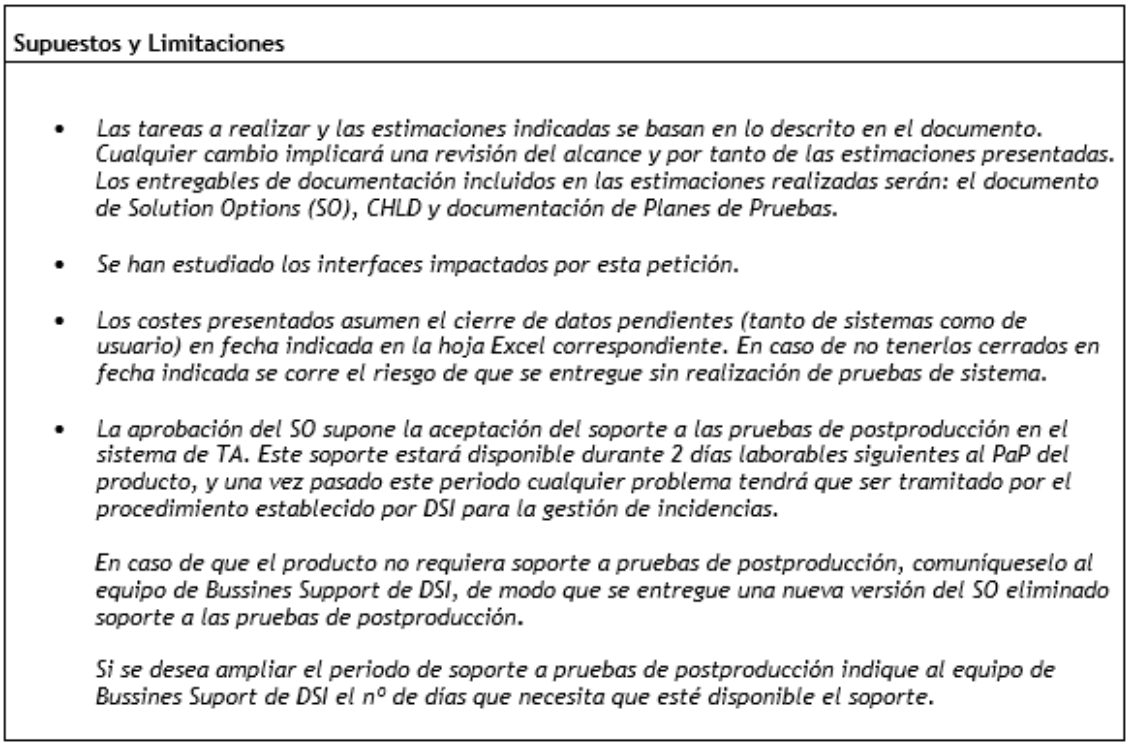


Contenido

Figura C.2: Estructura Documento Actividad de Exploración de Conceptos

A continuación se define la descripción del proyecto que, como se ha mencionado anteriormente, describe el alcance del proyecto, la definición funcional, las áreas involucradas, una planificación de las actividades a realizar y el coste que le supondrá al cliente.

Por último podemos ver un apartado llamado Supuestos y Limitaciones. En él se encuentra una descripción, Figura C.3, de otros aspectos que pueden afectar al coste del proyecto y cómo se actuará en dichos casos.



Supuestos y Limitaciones

- *Las tareas a realizar y las estimaciones indicadas se basan en lo descrito en el documento. Cualquier cambio implicará una revisión del alcance y por tanto de las estimaciones presentadas. Los entregables de documentación incluidos en las estimaciones realizadas serán: el documento de Solution Options (SO), CHLD y documentación de Planes de Pruebas.*
- *Se han estudiado los interfaces impactados por esta petición.*
- *Los costes presentados asumen el cierre de datos pendientes (tanto de sistemas como de usuario) en fecha indicada en la hoja Excel correspondiente. En caso de no tenerlos cerrados en fecha indicada se corre el riesgo de que se entregue sin realización de pruebas de sistema.*
- *La aprobación del SO supone la aceptación del soporte a las pruebas de postproducción en el sistema de TA. Este soporte estará disponible durante 2 días laborables siguientes al PaP del producto, y una vez pasado este periodo cualquier problema tendrá que ser tramitado por el procedimiento establecido por DSI para la gestión de incidencias.*

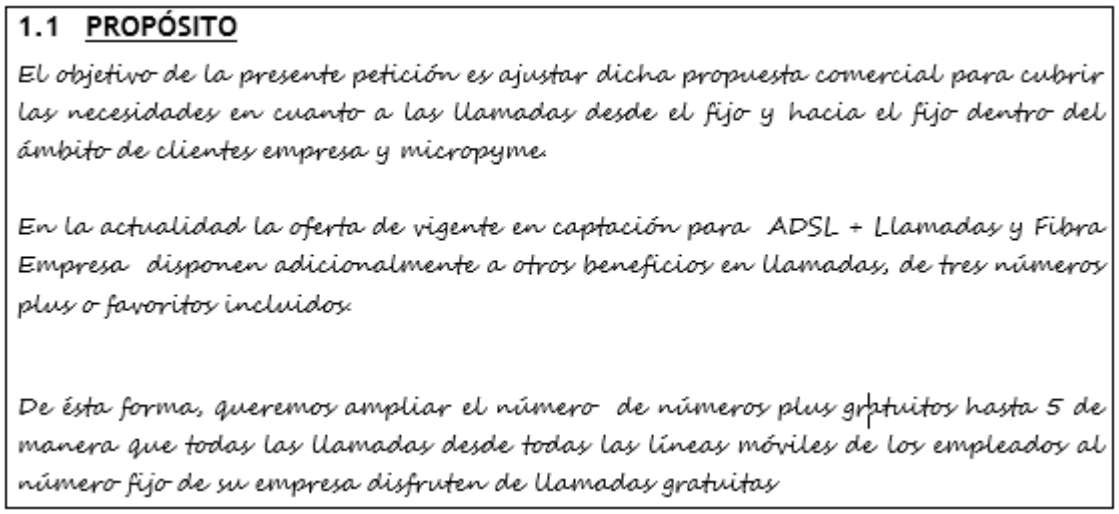
  En caso de que el producto no requiera soporte a pruebas de postproducción, comuníqueselo al equipo de Bussines Support de DSI, de modo que se entregue una nueva versión del SO eliminado soporte a las pruebas de postproducción.

  Si se desea ampliar el periodo de soporte a pruebas de postproducción indique al equipo de Bussines Suport de DSI el nº de días que necesita que esté disponible el soporte.

Figura C.3: Supuestos y Limitaciones del Documento Actividad de Exploración de Conceptos

C.3. Documento Actividad de Especificación de Requisitos

Una vez se han definido correctamente la implementación que se desea realizar, se especifican los requisitos funcionales y no funcionales, redactando un documento donde quedan correctamente definidos. En primer lugar se aclara el propósito del proyecto, Figura C.4, para más adelante, como se puede ver en el esquema del Documento de Especificación de Requisitos, Figura C.5, definir los requisitos agrupándolos por tipos.



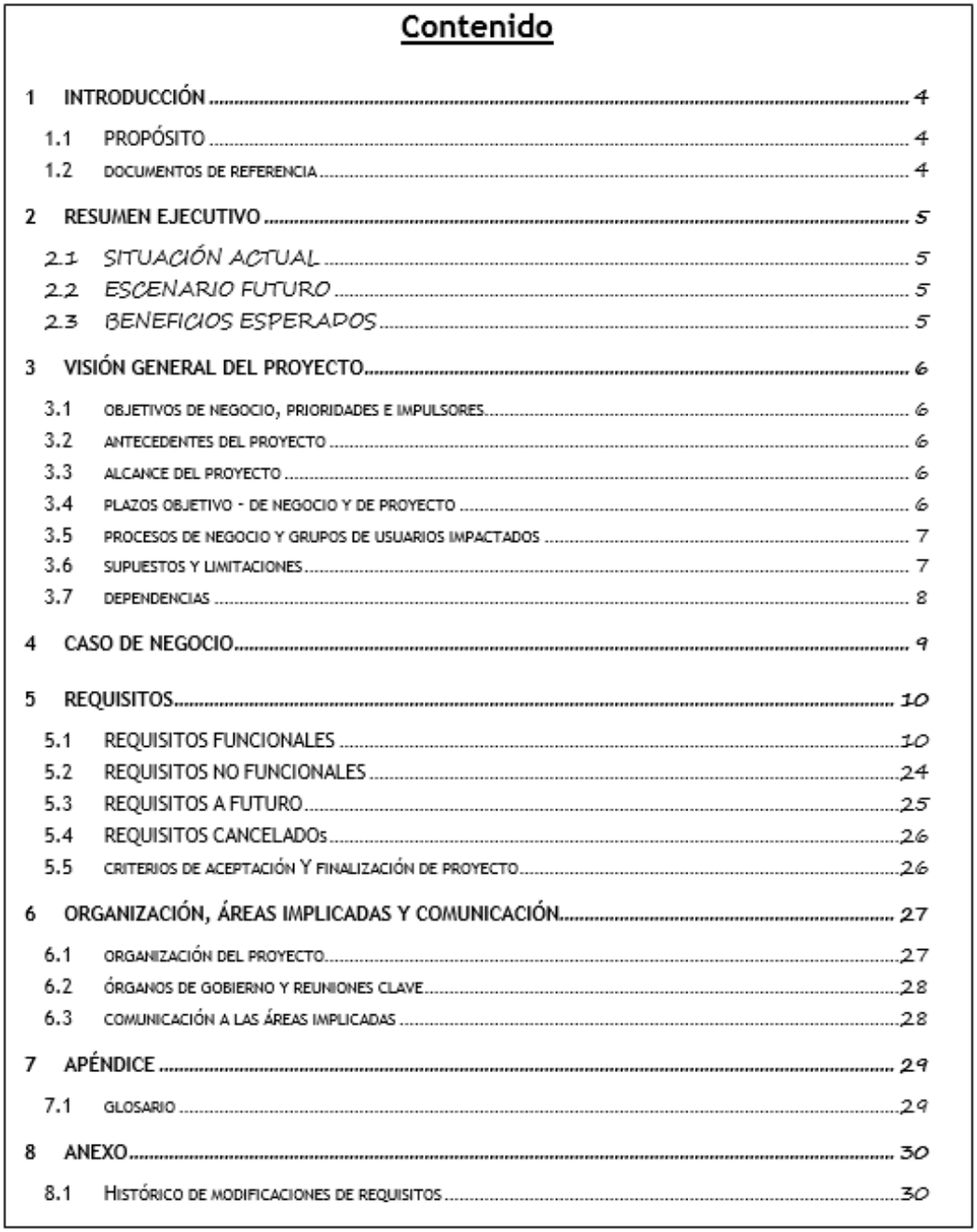
1.1 PROPÓSITO

El objetivo de la presente petición es ajustar dicha propuesta comercial para cubrir las necesidades en cuanto a las llamadas desde el fijo y hacia el fijo dentro del ámbito de clientes empresa y micropyme.

En la actualidad la oferta de vigente en captación para ADSL + Llamadas y Fibra Empresa disponen adicionalmente a otros beneficios en llamadas, de tres números plus o favoritos incluidos.

De ésta forma, queremos ampliar el número de números plus gratuitos hasta 5 de manera que todas las llamadas desde todas las líneas móviles de los empleados al número fijo de su empresa disfruten de llamadas gratuitas

Figura C.4: Apartado Propósito del Documento Especificación de Requisitos



Contenido

Figura C.5: Esquema Documento Especificación de Requisitos

La Tabla 3.2 muestra cómo se definen los requisitos, colocándolos en una tabla donde se muestra su nombre, una descripción, la fecha en la que se añadieron, el área al que afectan y la prioridad.

Este documento es fundamental ya que sirve de guía en las actividades de diseño e implementación.

C.4. Plan de Pruebas

Una vez se ha llevado a cabo la implementación, se procede con la ejecución de las pruebas. Para asegurar que se realizan las pruebas adecuadas y así garantizar el buen funcionamiento del sistema desarrollado, se crea un plan de pruebas donde se definen todas las pruebas que se han de validar. La Tabla 3.3 contiene un extracto de un caso del plan de pruebas, donde se puede ver alguno de los campos que lo componen, como la descripción o el resultado esperado.

A continuación se describen todos los campos que componen cada uno de los casos de prueba:

- ID del caso.
- Título del caso.
- Entorno.
- Agrupación funcional.
- Requisito que se comprueba.
- Descripción.
- Resultado esperado.
- Tiempo estimado de ejecución.
- Área.
- Estado de ejecución.

C.5. Sistema Evaluación de Pruebas

En este proyecto el cliente debe validar las pruebas, por lo que se utiliza un sistema de evaluación de pruebas, donde los usuarios deben adjuntar las evidencias necesarias a cada caso con el fin de obtener la validación del cliente. La Figura C.6 muestra un caso ya validado por el usuario.

Figura C.6: Caso en Estado *Passed* en el Sistema Evaluación de Pruebas

Cuando todos los casos se encuentran en estado *passed* se da por finalizada la actividad de pruebas y se sube a un entorno productivo.

C.6. Correos Electrónicos Relevantes

C.6.1. Correo Electrónico C.1

De: LopezBartolome, C.
Para: TelcoO.CatalogoFijo
Cc: SQAFijo
Asunto: [PA 5 Números Plus] No se muestran los PA esperados

Buenos días,
Hemos podido avanzar el alta de la oferta ABPLU y obtener los PA. Sin embargo las ofertas que devuelve catálogo no son las esperadas:

- No se debería ofrecer el antiguo Producto sino el nuevo PA Promoción 5 Números Plus gratis (Empresas), este último como obligatorio y desmarcado.
- Por otro lado, el PABT02 (Números Plus Empresa) aparece correctamente como opcional, pero no aparece desmarcado.

¿Podéis revisarlo?

Muchas gracias.

-Traza llamada a catálogo:

13.01.2014 11:03:35 [CallDispatcher] [usu475153/v95STgSJ15v277]: Formada la URL de llamada = http://XXX.24.XXX.168:XXXX/invoke/u2.Dispatcher/dispatcher?codWDispatcher=DIC26&herramienta=xnet.edv&loginUsuario=usu475153&YAX_LANG=esp&distriCode=475153&fecha=&idServicioPS=&tipoCliente=2&marca=ORANGE&canalVenta=XX&xmlData=<![CDATA[<?xml version="1.0" encoding="ISO-8859-1"?><filtros><descrFiltro>tipoProvision</descrFiltro><valorFiltro></valorFiltro></filtros><filtros><descrFiltro>tipoCliente</descrFiltro><valorFiltro>CASIMO</valorFiltro></filtros><filtros><descrFiltro>friend</descrFiltro><valorFiltro>N</valorFiltro></filtros>]]>&tipoOperacion=1&login=raulgomez&telefono=&offerName=ABPLU&operadorRed=5&idProvision=&codigoPromocion=8248
13.01.2014 11:03:52 [CallDispatcher] [usu475153/v95STgSJ15v277]: Tiempo llamada DICXX : 16690 ms

_____
Carlos López Bartolomé.

C.6.2. Correo Electrónico C.2

De: i. @desarrollo.com
Para: LopezBartolome, C.
Cc: SQAFijo
Asunto: RE: [PA 5 Números Plus] No se muestran los PA esperados

Hola,

En el proyecto no habla nada de desmarcar el Producto anterior, el proyecto solo es una nueva promoción, en este caso la promo 2939 y asociación a ofertas.

Un saludo,

Israel
+34 91 xxx xx xx
i. @desarrollo.com

_____
Carlos López Bartolomé.

C.6.3. Correo Electrónico C.3

De: LopezBartolome, C.
Para: TelcoO.CatalogoFijo
Cc: SQAFijo
Asunto: [PA 5 Números Plus] Las pruebas no se ajustan a los requisitos

Buenos días,

La prueba 1 descrita en el plan de pruebas del proyecto Nuevo PA Promo 5 Num Plus gratis (Empresas) no se ajusta a los requisitos de producto, según nos ha comentado desarrollo. En concreto, en los requisitos solo se indica que aparezca como opcional el PA 5 números plus, mientras que el plan de pruebas especifica que aparezca como obligatorio y desmarcado en lugar del Producto anterior.

¿Podrías confirmarnos esto?
Muchas gracias.

_____
Carlos López Bartolomé.

C.6.4. Correo Electrónico C.4

De: LopezBartolome, C.
Para: Laura.jefeEquipo@accenture.com
Cc:
Asunto: [PA 5 Números Plus] Informe diario

Buenos días,

- Ejecutando la prueba 1 se ha encontrado un problema, ya que en los requisitos solo se indica que aparezca como opcional el PA 5 números plus.

  Debido a esto sigue apareciendo el Producto anterior. Esto difiere de lo especificado en el plan de pruebas.

- Se ha escrito al usuario para comentárselo.

Nombre del Proyecto: Nuevo PA Promo 5 Num Plus gratis (Empresas)	**Cumplimiento Planificación Día Previo: NO**
Camino Libre Planificado Día Siguiente: **0 casos**	

PUNTOS DE BLOQUEO Y PLAN DE ACCIÓN	**Descripción**	**Grado de Bloqueo TOTAL**	**Grado de Bloqueo PARCIAL**	**Estado**
proyecto Nuevo PA Promo 5 Num Plus gratis (Empresas)	Las pruebas no se ajustan a los requisitos del proyecto	100%	100%	Abierto

Un saludo,

_____

Carlos López Bartolomé.

C.6.5. Correo Electrónico C.5

De: Laura.jefeEquipo@accenture.com
Para: TelcoO.CatalogoFijo; i.@desarrollo.com
CC: c.lopezbartolome@accenture.com;
Asunto: RE: URGENTE: [PA 5 Números Plus] Cambio de alcance

Hola,

Tras la conferencia que tuvimos ayer, en la que hablamos que necesitábais una nueva solución para el proyecto:

Cambiar el actual Producto anterior :

- Cambiar la descripción a "5 números plus gratuitos" en todos los sistemas
- Que ahora permita añadir 5 números favoritos en vez de 3 como ahora en todos los sistemas, tanto para Altas, Cambios de ofertas y Postventa.

He hablado con los sistemas de Accenture afectados y desde Facturación me comentan que si tienen que cambiar el componente y que es correcto como estaba desplegado, les añado en copia.

De todas formas se puede realizar el cambio y hemos realizado las estimaciones de esfuerzo pertinentes:

Catálogo:	2 jornadas
OE:	1 jornada
Facturación:	3 jornadas
PS:	2 jornadas
Portales:	3 jornadas

Cualquier comentario o duda decidnos.

Un saludo,

Laura Jefe de equipo
Accenture

Anexo D. Bitácora de Double Commander

D.1. Descripción

Figura D.1: Logo Double Commander

DoubleCommander, cuyo logo se muestra en la Figura D.1, es un explorador de archivos a doble pantalla (Figura D.2), que permite mover archivos de un directorio a otro sin ninguna complicación. Además tiene un gran número de acciones complementarias que se pueden realizar y que hacen de DoubleCommander una herramienta bastante completa y sencilla al mismo tiempo.

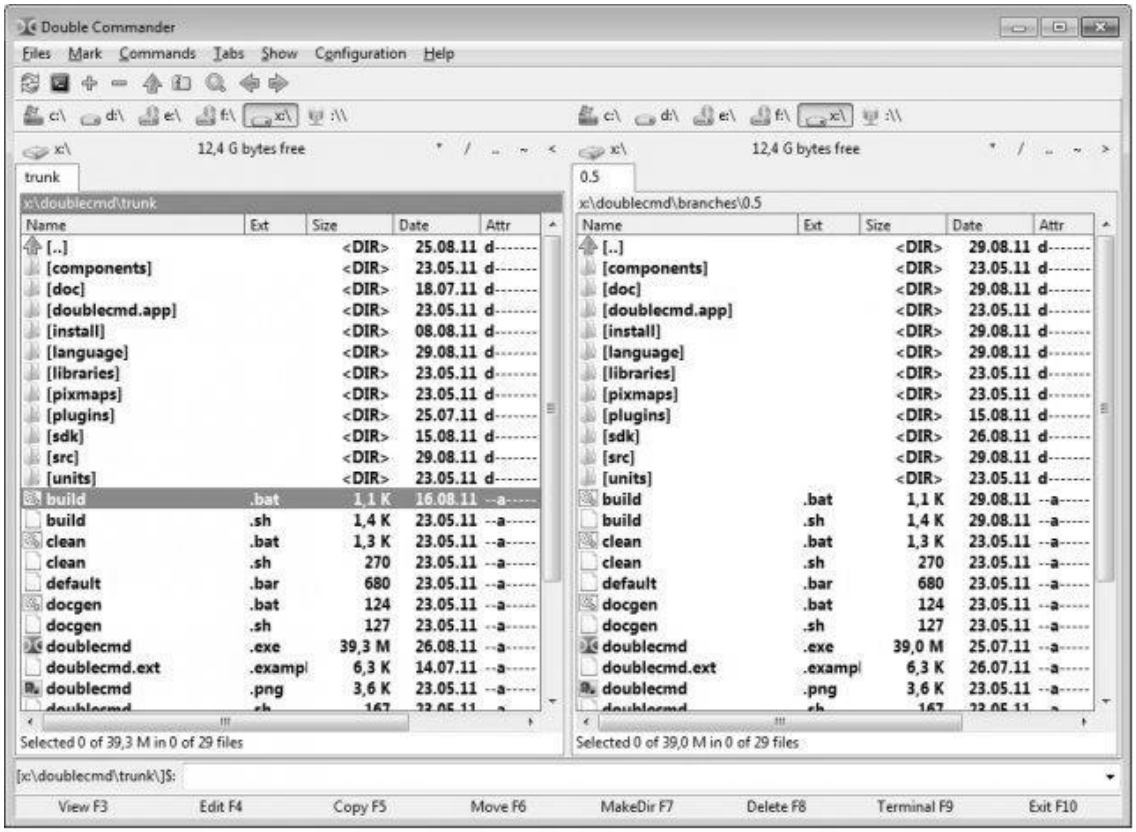


Figura D.2: Ejemplo Pantalla Principal Double Commander

Este proyecto se encontró en la página de OSS, doublecmd.sourceforge.net, en la que se puede encontrar toda la documentación existente, el foro de la aplicación, las listas de correo y los enlaces para realizar las descargas tanto de la aplicación como del código fuente.

D.2. Herramientas

A continuación se describen las herramientas que se han utilizado durante la colaboración en este proyecto.

D.2.1. Mantis

Mantis Bug Tracker, www.mantisbt.org, es un software que constituye una solución completa para gestionar tareas en un equipo de trabajo. Es una aplicación *open source* realizada con PHP y MYSQL que destaca por su facilidad y flexibilidad de instalar y configurar. Esta aplicación se utiliza para testear soluciones, hacer un registro histórico de alteraciones y gestionar equipos remotamente.

Esta aplicación fue empleada para reportar o proponer una nueva funcionalidad, que consistía en implementar un botón que permitiese volver al explorador de archivos, manteniendo el directorio en el que se estaba navegando. La Figura D.3 muestra un extracto de la pantalla de creación de un *Bug*.

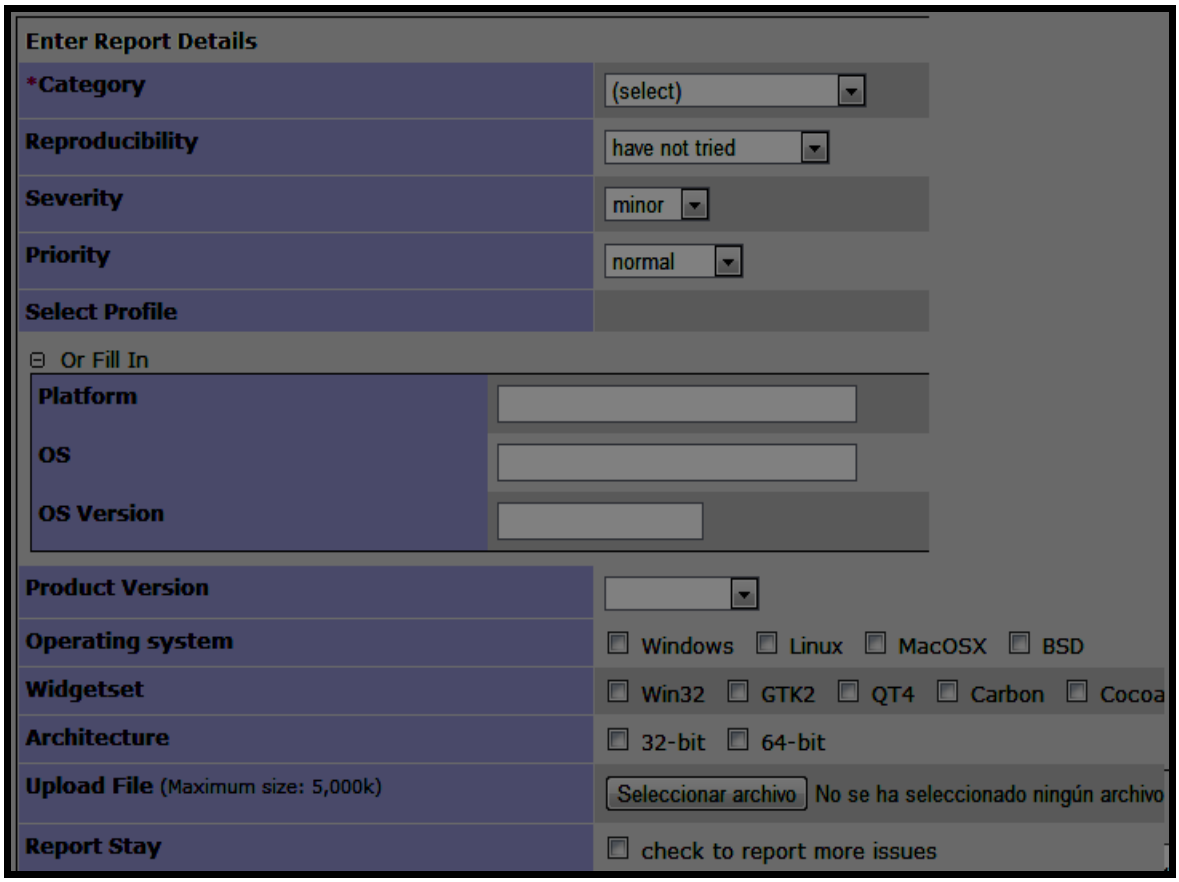


Figura D.3: Estracto Pantalla Creación *Bug* en Mantis

Una vez se ha realizado una aportación, ésta pasa por los siguientes estados:

- **New**: Nueva propuesta/bug reportado.

- **Feedback**: Encontramos esto en errores reportados que ya han sido solucionados en versiones siguientes del producto.
- **Acknowledged**: Significa que el *bug* ha sido admitido como tal pero no tienen pensado arreglarlo en el presente.
- **Confirmed**: Confirmado el error a falta de asignarlo.
- **Assigned**: Error asignado.
- **Resolved**: El error/propuesta reportado ya ha sido resuelto.
- **Closed**: El *bug* ha sido cerrado por invalidez.

D.3. Comunicación con la Comunidad

A continuación se muestran las comunicaciones más relevantes llevadas a cabo con la comunidad OSS implicada.

D.3.1. Propuesta Nueva Funcionalidad

Original: *When scrolling through the files, there is the option to click a button that opens the VFS List to see networks. The problem is that once you've stopped seeing these networks, you can not go to the directory in which it was before. In many cases this can be a big problem, because if you have navigated much in directories, back to the directory you were working can be very complicated.*

My proposal is a button that allows return to pre-finish the list of networks directory.

Thank you for your time.

Regards,

Carlos

Castellano: A la hora de desplazarse a través de los archivos, existe la opción de pulsar en un botón que abre la VFS List para ver redes. El problema es que una vez has dejado de ver dichas redes, no se puede volver al directorio en el que se estaba antes. En muchos casos esto puede ser un gran inconveniente, ya que si se ha navegado mucho en los directorios, volver al directorio en el que se estaba trabajando puede ser muy complicado.

Mi propuesta consiste en un botón que permita volver al directorio previo al terminar de ver la lista de redes.

Muchas gracias por su tiempo.

Un saludo,

Carlos

D.3.2. Contestación de la comunidad

Original: *The "button" already exists. It's directory history. It can be called by mouse click on path or by assign hotkey to ViewHistory command.*

Castellano: Dicho "botón" ya existe, es el historial de directorios. Se puede acceder mediante un click o usando el comando *ViewHistory command*.

Anexo E. Bitácora de FileZilla

E.1. Descripción

Figura E.1: Logo FileZilla

FileZilla, cuyo logo se muestra en la Figura E.1, es un cliente FTP multiplataforma de código abierto, que permite a un usuario crear una lista de sitios FTP con sus datos de conexión, como el número de puerto a usar, o si se utiliza inicio de sesión normal o anónima. Es un sistema de doble pantalla, Figura E.2, que permite navegar por las carpetas, ver y alterar sus contenidos tanto en la máquina local como en la remota, utilizando una interfaz de tipo árbol de exploración y pudiendo arrastrar y soltar archivos entre los ordenadores local y remoto.

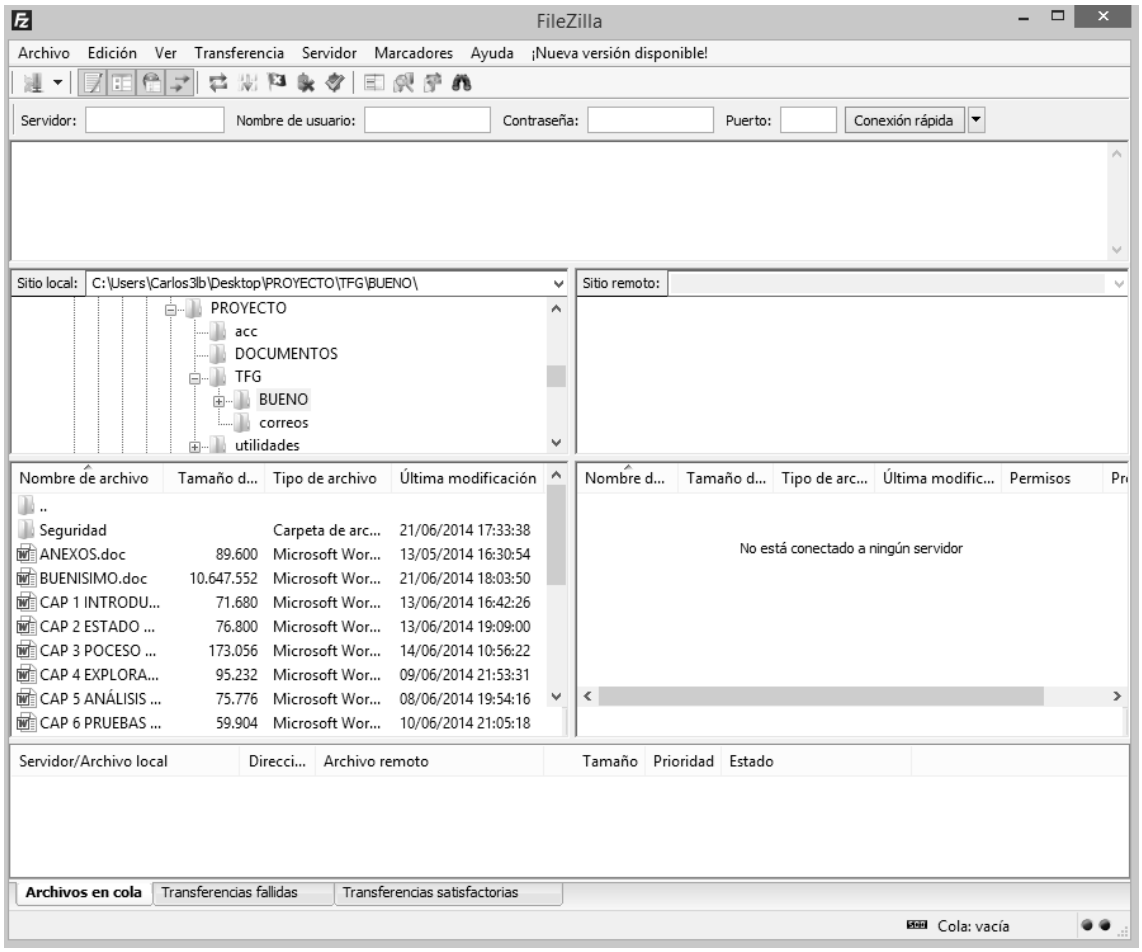


Figura E.2: Ejemplo Pantalla FileZilla

Este proyecto se eligió al ser una aplicación muy usada en mi trabajo. Para acceder a la comunidad se debe usar la página del proyecto, filezilla-project.org. En ella se encontró un sistema para la gestión de errores y aportaciones basado en tickets y un sistema de actualización de traducciones. También se puede descargar la aplicación y su código.

Para poder ejecutar el código, FileZilla cuenta con una página. https://wiki.filezilla-project.org/Compiling_FileZilla_3_under_Windows, donde se describe de forma muy clara los pasos a seguir.

E.2. Herramientas

A continuación se describen las herramientas que se han utilizado durante la colaboración en este Proyecto.

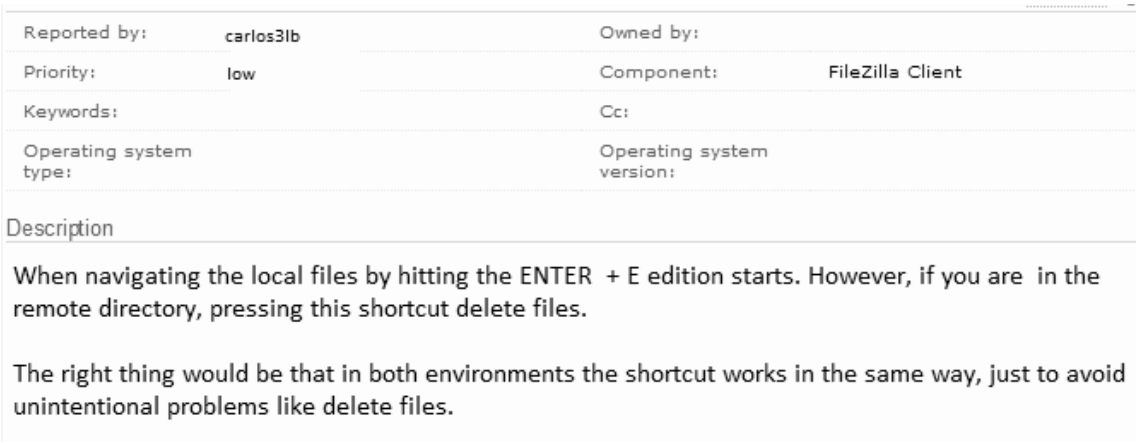
E.2.1. Sistema de Tickets de FileZilla

La página cuenta con un sistema para la creación de Tickets, http://trac.filezilla-project.org/wiki/Queries, que sirve para reportar fallos, proponer nuevas funcionalidades y ver los parches subidos, dividido en los diferentes módulos que forman la aplicación, Figura E.3.

	Open	Pending further information	Closed	Search
FileZilla Client:	Bug reports (467) Feature requests (730) Patches (24)	Bug reports (27) Feature requests (1) Patches (7)	Bug reports (2413) Feature requests (1001) Patches (87)	Summary: Description: Search
FileZilla Server:	Bug reports (71) Feature requests (138) Patches (0)	Bug reports (4) Feature requests (0) Patches (1)	Bug reports (231) Feature requests (164) Patches (16)	Summary: Description: Search
Other:	Bug reports (4) Feature requests (8) Patches (1)	Bug reports (1) Feature requests (0) Patches (0)	Bug reports (1284) Feature requests (344) Patches (46)	Summary: Description: Search

Figura E.3: Sistema Gestión Tickets FileZilla

Al navegar por la herramienta se puede buscar tickets por tema, ordenar por fecha, añadir comentarios a tickets existentes y crear nuevos. La Figura E.4 muestra la descripción del tiquet creado para reportar el error que se detectó y que consitía en que a la hora de navegar por el directorio local, el comando E + ENTER abría un archivo para editarlo, mientras que en remoto, ese mismo comando eliminaba archivos y ficheros.



Reported by:	carlos3lb	Owned by:	
Priority:	low	Component:	FileZilla Client
Keywords:		Cc:	
Operating system type:		Operating system version:	

Description

When navigating the local files by hitting the ENTER + E edition starts. However, if you are in the remote directory, pressing this shortcut delete files.

The right thing would be that in both environments the shortcut works in the same way, just to avoid unintentional problems like delete files.

Figura E.4: Ticket Bug Atajo para Editar en Local y Borrar en Remoto

Una vez creado el ticket, los otros usuarios pueden comentar y los administradores pueden gestionar la entrada. En este caso la eliminaron y la crearon en futuros cambios, ya que yo la había creado en el sitio incorrecto.

E.2.2. Sistema de Actualización de Traducciones de FileZilla

Filezilla es una aplicación que se encuentra traducida a un gran número de lenguas. Esto es posible gracias a lo fácil que es aportar una traducción o mejorarla.

En la página de traducciones, https://filezilla-project.org/translations.php, se encuentra una descripción de los pasos a seguir para colaborar con una traducción. Para ello simplemente hay que, en primer lugar, descargarse el fichero .po correspondiente al idioma que se desea traducir y editar las palabras o frases que no tengan traducción, Figura E.5.

```
7322  #: ../../source/FileZilla3/locales/../src/interface/resources/xrc/update.xrc:118
7323  #, fuzzy
7324  msgid "The new version could not be downloaded, please retry later."
7325  msgstr "El servidor no pudo ser añadido."
7326
7327  #: ../../source/FileZilla3/locales/../src/interface/resources/xrc/update.xrc:105
7328  msgid "The new version has been saved in your Downloads directory."
7329  msgstr "La nueva version ha sido guardada en su directorio descargas."
7330
7331  #: ../../source/FileZilla3/locales/../src/interface/QueueView.cpp:1968
7332  msgid "The queue will not be saved."
7333  msgstr "La cola no será guardada."
```

Figura E.5: Ejemplo Traducción al Castellano para FileZilla

Una vez terminada la traducción, se debe compilar usando el compilador online que ofrece la página y probarlo con cualquiera de las versiones existentes. Una vez comprobado se envía por correo a un usuario que lo comprobará y actualizará.

En la página traducciones aparece una tabla con los estados de las traducciones. La Figura E.6 muestra el estado de las traducciones antes de nuestra colaboración y la Figura E.7 es el de la siguiente actualización. Como se puede comprobar el número de

traducciones en español ha aumentado, pero el porcentaje de traducciones completadas baja debido a la inserción de nuevas frases sin traducir en la nueva versión.

Language	Translated		Fuzzy		Untranslated		Graph	Download	Binary
Aragonese (an)	1551	(96.4%)	39	(2.5%)	17	(1.1%)		an.po	an.mo
Arabic (ar)	1589	(98.8%)	14	(0.9%)	4	(0.3%)		ar.po	ar.mo
Bulgarian (Bulgaria) (bg_BG)	1588	(98.7%)	14	(0.9%)	5	(0.4%)		bg_BG.po	bg_BG.mo
Catalan (ca_ES@valencia)	1320	(82.0%)	177	(11.1%)	110	(6.9%)		ca_ES@....po	ca_ES@....mo
Catalan (ca)	1320	(82.0%)	177	(11.1%)	110	(6.9%)		ca.po	ca.mo
Corsican (co)	1564	(97.2%)	36	(2.3%)	7	(0.5%)		co.po	co.mo
Czech (Czech Republic) (cs_CZ)	1555	(96.7%)	36	(2.3%)	16	(1.0%)		cs_CZ.po	cs_CZ.mo
Danish (Denmark) (da_DK)	1587	(98.7%)	16	(1.0%)	4	(0.3%)		da_DK.po	da_DK.mo
German (de)	1601	(99.5%)	4	(0.3%)	2	(0.2%)		de.po	de.mo
Greek (el)	1601	(99.6%)	3	(0.2%)	3	(0.2%)		el.po	el.mo
Spanish (es)	1512	(94.0%)	78	(4.9%)	17	(1.1%)		es.po	es.mo

Figura E.6: Tabla Estado Traducciones FileZilla Anterior a Nuestra Colaboración

Language	Translated		Fuzzy		Untranslated		Graph	Download	Binary
Aragonese (an)	1551	(93.3%)	64	(3.9%)	45	(2.8%)		an.po	an.mo
Arabic (ar)	1607	(96.7%)	25	(1.6%)	28	(1.7%)		ar.po	ar.mo
Bulgarian (Bulgaria) (bg_BG)	1588	(95.6%)	39	(2.4%)	33	(2.0%)		bg_BG.po	bg_BG.mo
Catalan (ca_ES@valencia)	1320	(79.4%)	202	(12.2%)	138	(8.4%)		ca_ES@....po	ca_ES@....mo
Catalan (ca)	1320	(79.4%)	202	(12.2%)	138	(8.4%)		ca.po	ca.mo
Corsican (co)	1564	(94.1%)	61	(3.7%)	35	(2.2%)		co.po	co.mo
Czech (Czech Republic) (cs_CZ)	1555	(93.6%)	61	(3.7%)	44	(2.7%)		cs_CZ.po	cs_CZ.mo
Danish (Denmark) (da_DK)	1587	(95.5%)	41	(2.5%)	32	(2.0%)		da_DK.po	da_DK.mo
German (de)	1601	(96.3%)	29	(1.8%)	30	(1.9%)		de.po	de.mo
Greek (el)	1613	(97.1%)	21	(1.3%)	26	(1.6%)		el.po	el.mo
Spanish (es)	1529	(92.0%)	103	(6.3%)	28	(1.7%)		es.po	es.mo

Figura E.7: Tabla Estado Traducciones FileZilla Posterior a Nuestra Colaboración

E.3. Comunicación con la Comunidad

A continuación se muestran las comunicaciones más relevantes llevadas a cabo con la comunidad OSS implicada.

E.3.1. Reporte del Error

Original: *When navigating the local files by hitting the ENTER + E edition starts. However, if you are in the remote directory, pressing this shortcut delete files.*

The right thing would be that in both environments the shortcut Works in the same way, just to avoid unintentional problems like delete files.

Castellano: A la hora de navegar por los archivos locales, pulsando E + ENTER se inicia la edición. Sin embargo, si se está navegando en el directorio remoto, al pulsar este atajo se borran los archivos.

Lo correcto sería que en ambos entornos el atajo funcionara igual, para evitar problemas como borrados involuntarios.

E.3.2. Contestación al Reporte del Error

Original: *The ticket has been removed and created again as Feature requests and is not considered an error, but an improvement..*

Castellano: El ticket ha sido eliminado y creado de nuevo en *Feature requests* ya que no se considera un error sino una mejora.

E.3.3. Solución del Error por otro Usuario

Original: *Modification:*

> *"E" is the shortcut to edit local and remote.*
> *"L" is now the shortcut to delete remote and local. ("D" is for Download).*
> *Diff is attached.*

Castellano: Modificación:

> "E" es el atajo para editar en remoto y local.
> "L" es ahora el atajo para borrar en remoto y local. ("D" is for Download).
> Se adjunta Diff.

Adjuntos: Código solución error, Figura E.8.

src/interface/resources/menus.xrc

```
292 292       <help>Enter selected directory</help>
293 293     </object>
294 294     <object class="wxMenuItem" name="ID_EDIT">
295         -   <label>&amp;View/Edit</label>
    295     +   <label>View/&amp;Edit</label>
296 296     </object>
297 297     <object class="separator"/>
298 298     <object class="wxMenuItem" name="ID_MKDIR">
... ...
308 308     </object>
309 309     <object class="separator"/>
310 310     <object class="wxMenuItem" name="ID_DELETE">
311         -   <label>D&amp;elete</label>
    311     +   <label>De&amp;lete</label>
312 312       <help>Delete selected files and directories</help>
313 313     </object>
314 314     <object class="wxMenuItem" name="ID_RENAME">
```

Figura E.8: Código Solución Atajos para Filezilla

E.3.4. Correo Electrónico Traducción al Castellano

Para: tim.kosse@filezilla-project.org
De: carlos3lb@hotmail.com
CC:
Asunto: Spanish FileZilla Translation

Original:

Hi!

This is the complete spanish translation for the FileZilla proyect.

Thank you very much for your work.

Best regards.

Carlos López

Enviado desde Correo de Windows

Castellano:

Hola!

Ésta es la traducción completa al Castellano para el Proyecto FileZilla.

Muchas gracias por vuestro trabajo.

Saludos.

Carlos López

Enviado desde Correo de Windows

Anexo F. Bitácora de VLC Media Player

F.1. Descripción

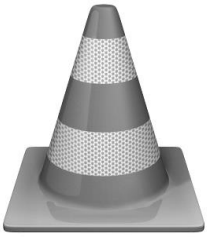


Figura F.1: Logo VLC Media Player

VLV Media player, cuyo logo se muestra en la Figura F.1, es un reproductor multimedia simple, rápido y potente (Figura F.2), que reproduce archivos, discos, cámaras de vídeo, dispositivos y flujos. En cuanto a los formatos de reproducción reproduce la mayoría de códecs sin necesidad de paquetes de códecs, como MPEG-2, DivX, H.264, MKV, WebM, WMV, MP3. Es multiplataforma y completamente libre, sin programas espía, sin anuncios y sin seguimiento del usuario.

Figura F.2: Ejemplo Reproducción en VLC Media Player

Este proyecto se eligió al ser una aplicación muy usada en mi día a día. Para descargar el programa se usó su página principal, http://www.videolan.org/vlc/. Para acceder a la comunidad existe la página https://wiki.videolan.org/Developers_Corner. En ella, Figura F.3, se encuentra toda la información para descargar el código y cuenta con dos medios para establecer contacto con la comunidad: una aplicación que funciona mediante tickets; y un sistema de listas de correo, que fue el que se decidió utilizar para comprobar su funcionamiento y efectividad.

VLC Developers Corner

(Redirected from Developers Corner)

Welcome to **VLC Developers Corner**,

This is a directory of everything to do with the development of VideoLAN's projects.

Check VLC's page to get info on VLC.

Coding on VLC

- Get started at coding on VLC
- Get the Source Code and Compile VLC
- You might be interested in our Mini Projects or Janitorial projects
- Read our Hacker's Guide
  - Introduction to VLC's core
  - Modules loading documentation
  - How to write a module
  - How VLC modules load
- Code documentation
- Code Conventions

VLC development links

- Trac list of bugs
- vlc-devel mailing list, mailing list.
- Translation information and stats
- Ohloh statistics
- Buildbot
- Nightly Builds
- Launchpad (recent bug changes) (all bugs)

Figura F.3: Página VLC *Developers Corner* de VLC Media Player

F.2. Herramientas

A continuación se describen las herramientas que se han utilizado durante la colaboración en este proyecto.

F.2.1. Sistema de Listas de Correo de VLC Media Player

El sistema utilizado para la comunicación con la comunidad de VLC Media Player ha sido el de Listas de Correo. Al acceder al sistema, encontramos un enlace, que nos muestra las listas de correo según su tema, Figura F.4.

List administrators, you can visit the list admin overview page to find the management interface for your list.

If you are having trouble using the lists, please contact mailman@videolan.org.

List	Description
Android	Development of VLC for Android
Apple-bugreport	VLC Bug reports sent by Apple
biTStream-devel	*[no description available]*
dvblast-devel	Mailing list for DVBlast developers
iOS	Development of VLC on iOS
libaacs-devel	Development around libaacs
libbdplus-devel	Development around libbdplus
libbluray-devel	Development around libbluray
libdca-devel	Mailing list for libdca and DTS decoding

Figura F.4: Estracto Página Lista de Correos de VLC Media Player

Las listas de correo esten gestionadas por Mailman, http://www.gnu.org/software/mailman/index.html, que es un software libre para la gestión de listas de correo electrónico.

Al pulsar en cualquiera de los temas que sean de interés se accederá a una página de registro para unirnos a la lista de correo elegida y así poder empezar a colaborar.

F.3. Comunicación con la Comunidad

F.3.1. Correo Electrónico Consulta Medio para Proponer una Funcionalidad

Para: mailman@videolan.org
De: carlos3lb@hotmail.com
CC:
Asunto: Consultation on a new feature

Original:

Hi,

I'm new to the community and I wanted to know if mailing lists are the appropriate channel to propose new functionalities

Thanks for your time

Carlos

Enviado desde Correo de Windows

Castellano:

Hola!

Soy nuevo en la comunidad y quería saber si las listas de correo son el medio adecuado para proponer nuevas funcionalidades

Gracias por su tiempo

Carlos
Enviado desde Correo de Windows

F.3.2. Correo Electrónico RE: Consulta Medio para Proponer una Funcionalidad

Para: carlos3lb@hotmail.com
De: mailman@videolan.org
CC:
Asunto: RE: Consultation on a new feature

Original:

Yes. You can propose functionalities using the proper list the subject of your proposal

Castellano:

Sí. Puedes proponer funcionalidades usando la lista adecuada al tema de tu propuesta

F.3.3. Correo Electrónico Propuesta Funcionalidad

Para: vlc-devel@videolan.org
De: calos3lb@hotmail.com
CC:
Asunto: Propuesta Funcionalidad

Original:

Hello,

I would propose a new functionality to VLC Media Player.

The functionality is that while playing a file, you can scan the directories from the player in a sidebar or in a menu and play them or add them to the play queue. The idea is not having to use the file browser of the operating system.

The menu bar or hide it should be possible to not interfere with reproduction.

I await your response.

Carlos

Enviado desde Correo de Windows

Castellano:

Hola,

Me gustaría proponer una nueva funcionalidad para VLC Media Player.

La funcionalidad consiste en que mientras se reproduce un archivo, se pueda explorar los directorios desde el reproductor en una barra lateral o en un menú y poder reproducirlos o añadirlos a la cola de reproducción. La idea es no tener que utilizar el explorador de archivos del sistema operativo.

La barra o menú debería poderse ocultar para no interferir con la reproducción.

Espero su respuesta.

Carlos

Enviado desde Correo de Windows

F.3.4. Correo Electrónico RE: Propuesta Funcionalidad

Para: carlos3lb@hotmail.com
De: vlc-devel@videolan.org
CC:
Asunto: RE: Propuesta Funcionalidad

Original:

Unless i have gotten you wrong, this already exists in the playlist menu i.e. the playing audio or video resumes into a smaller screen at the bottom left, you see playlist on right and in left pane i.e. above the smaller audio / video, there is file system browse section where you can browse and add any new file while the main one keeps running until you play the new one.

Castellano:

A menos que te haya entendido mal, dicha funcionalidad ya existe en el menú de lista de reproducción. Al pulsar el video se mueve en una pantalla más pequeña y se puede ver una nueva pantalla con un explorador del sistema donde se puede navegar y agregar cualquier nuevo archivo, manteniendo el actual funcionando hasta que se seleccione otro.

F.3.5. Correo Electrónico RE2: Propuesta Funcionalidad

Para: vlc-devel@videolan.org
De: carlos3lb@hotmail.com
CC:
Asunto: RE2: Propuesta Funcionalidad

Original:

Well I understand what you meant but where you can navigate is through your playlist (you must have created them before) and what I mean is navigate through your computer files i.e. the default windows explorer but in one side of the player.

Maybe i am wrong and missundertood you.

Carlos

Enviado desde Correo de Windows

Castellano:

Bueno, entiendo lo que me quieres decir, pero donde se puede navegar es a través de la lista de reproducción (debe haber creado antes) y lo que propongo es navegar a través de los archivos del ordenador, es decir, el explorador de Windows por defecto, pero en un lado del reproductor.

Tal vez estoy equivocado y no te he entendido.

Anexo G. Bitácora de GanttProject

G.1. Descripción

Figura G.1: Logo GanttProject

GanttProject (Figura G.1) es una herramienta que como su propio nombre indica su funcionalidad es la de crear diagramas de Gantt, tal como muestra la Figura G.2. Además también crea diagramas de Pert y puedes gestionar proyectos asignando recursos ya sean humanos o materiales a las diferentes tareas del proyecto.

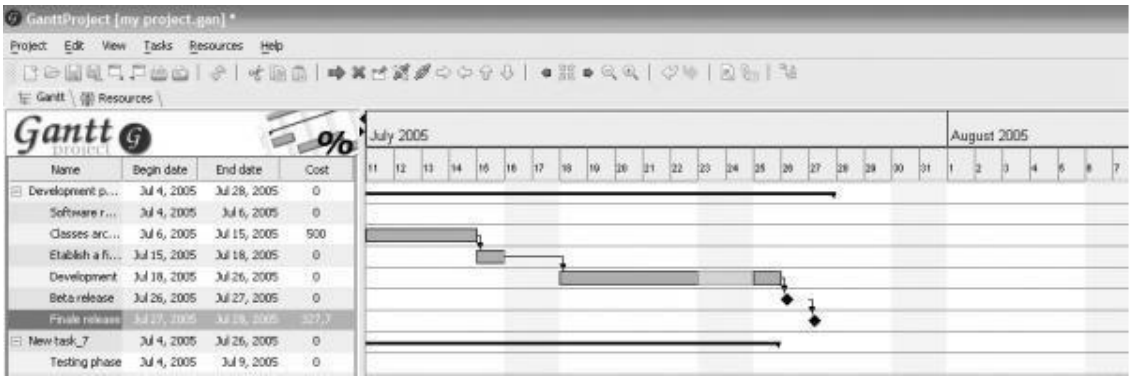


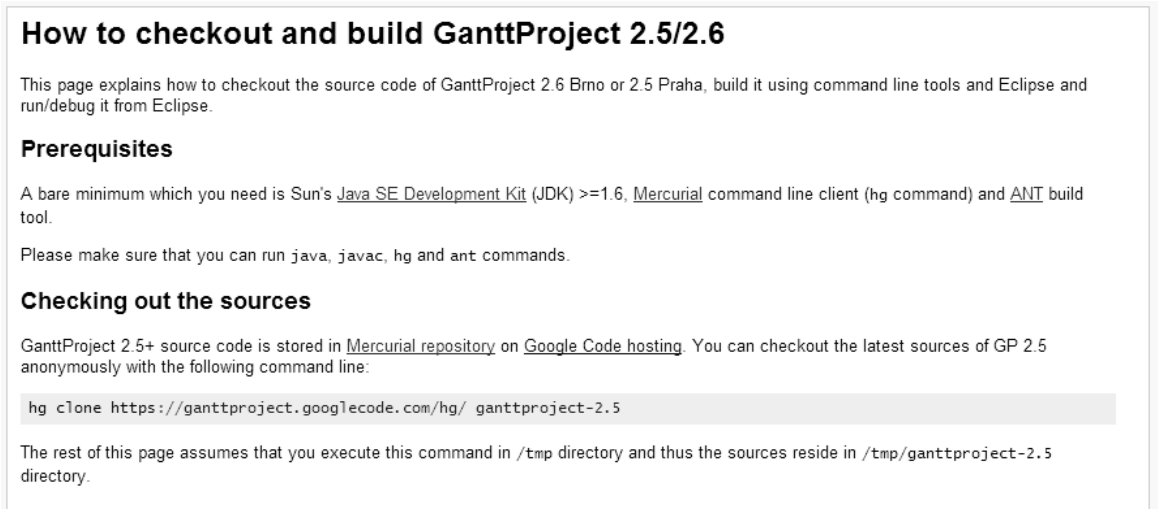
Figura G.2: Ejemplo de Uso de GanttProject

Este proyecto fue elegido ya que hemos usado programas parecidos en la universidad como por ejemplo Microsoft Project y pareció positivo conocer un programa OSS similar y que además contaba con numerosas referencias en cuanto a su facilidad de uso.

En la página del proyecto, http://www.ganttproject.biz/participate, se explican todos los tipos de colaboraciones que se pueden realizar, como donaciones, pruebas y reporte de errores, traducciones y desarrollo. Al acceder a la página de desarrollo, se trató de buscar una oportunidad solucionando un error, por lo que se accedió a la sección de errores, donde se encontró un mensaje, Figura G.3, donde se explicaba que ahora se utilizaba el sistema Google Code Pages para reportar errores.

Figura G.3: Mensaje en la Página Isues de GanttProject

En la página del proyecto se encuentra también un manual, Figura G.4, de cómo descargar y compilar GanttProject, y que fue de gran utilidad para el desarrollo.



How to checkout and build GanttProject 2.5/2.6

This page explains how to checkout the source code of GanttProject 2.6 Brno or 2.5 Praha, build it using command line tools and Eclipse and run/debug it from Eclipse.

Prerequisites

A bare minimum which you need is Sun's Java SE Development Kit (JDK) >=1.6, Mercurial command line client (`hg` command) and ANT build tool.

Please make sure that you can run `java`, `javac`, `hg` and `ant` commands.

Checking out the sources

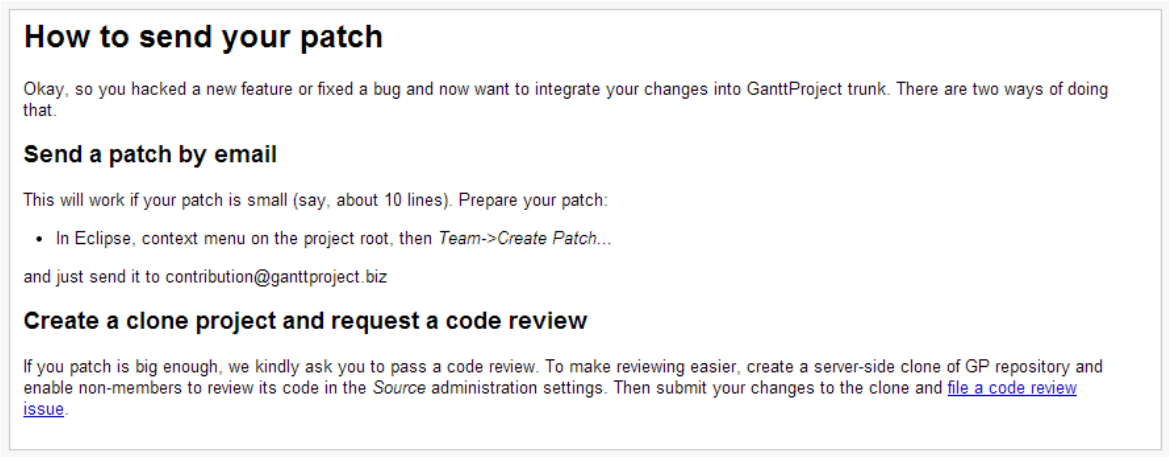
GanttProject 2.5+ source code is stored in Mercurial repository on Google Code hosting. You can checkout the latest sources of GP 2.5 anonymously with the following command line:

```
hg clone https://ganttproject.googlecode.com/hg/ ganttproject-2.5
```

The rest of this page assumes that you execute this command in `/tmp` directory and thus the sources reside in `/tmp/ganttproject-2.5` directory.

Figura G.4: Estracto Manual de Compilación de GanttProject

Una vez realizada la implementación, se pensó en utilizar la herramienta Mercurial, pero por si acaso se preguntó a la comunidad, que nos respondió que mejor por correo. Además, se leyó en la página, Figura G.5, que si no era un cambio grande, lo mejor era utilizar el correo electrónico.



How to send your patch

Okay, so you hacked a new feature or fixed a bug and now want to integrate your changes into GanttProject trunk. There are two ways of doing that.

Send a patch by email

This will work if your patch is small (say, about 10 lines). Prepare your patch:

- In Eclipse, context menu on the project root, then *Team->Create Patch...*

and just send it to contribution@ganttproject.biz

Create a clone project and request a code review

If you patch is big enough, we kindly ask you to pass a code review. To make reviewing easier, create a server-side clone of GP repository and enable non-members to review its code in the *Source* administration settings. Then submit your changes to the clone and file a code review issue.

Figura G.5: Manual de Cómo Subir Código de GanttProject

G.2. Herramientas

A continuación se describen las herramientas que se han utilizado durante la colaboración en este proyecto.

G.2.1. Google Code

Figura G.6: Logo Google Code

Google Code (Figura G.6) es un sitio de Google para desarrolladores interesados en el desarrollo OSS. El sitio contiene códigos fuente abiertos y una lista de sus servicios de apoyo público como el que hemos utilizado. GanttProject utiliza Google Code para gestionar las colaboraciones de los usuarios, https://code.google.com/p/ganttproject/issues/list. La Figura G.7 muestra la pantalla principal de la sección errores.

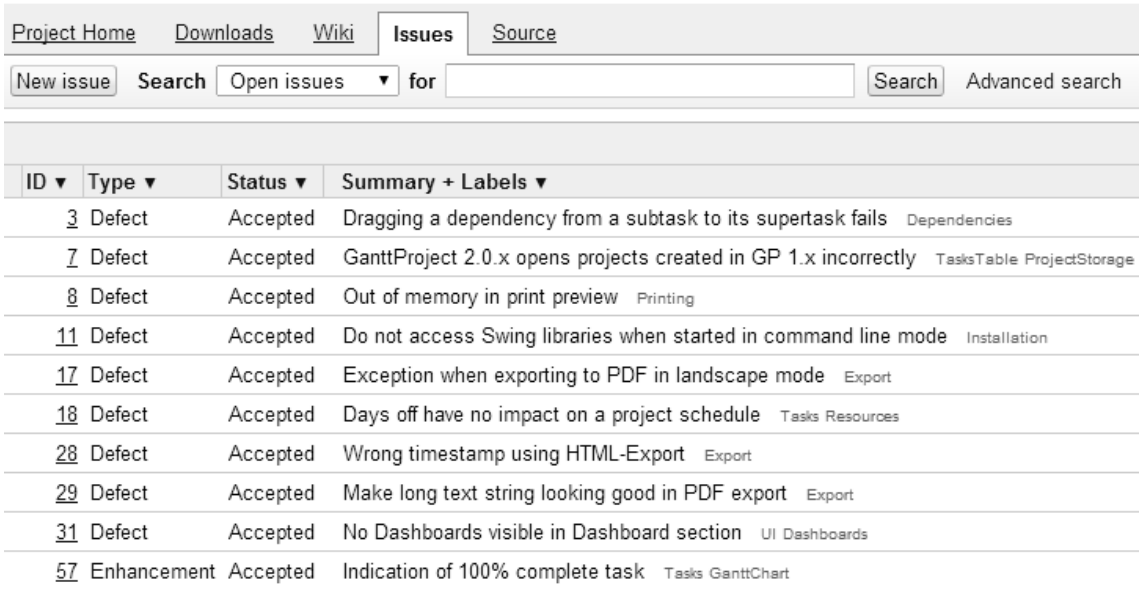


Figura G.7: Estracto Página principal de Gestión de Errores de GanttProject en Google Code

Para reportar un error, no es necesario estar registrado en la comunidad, basta con tener una cuenta de Google. La Figura G.8 muestra la pantalla de creación de un error, con la plantilla por defecto que se añade al iniciar un reporte.

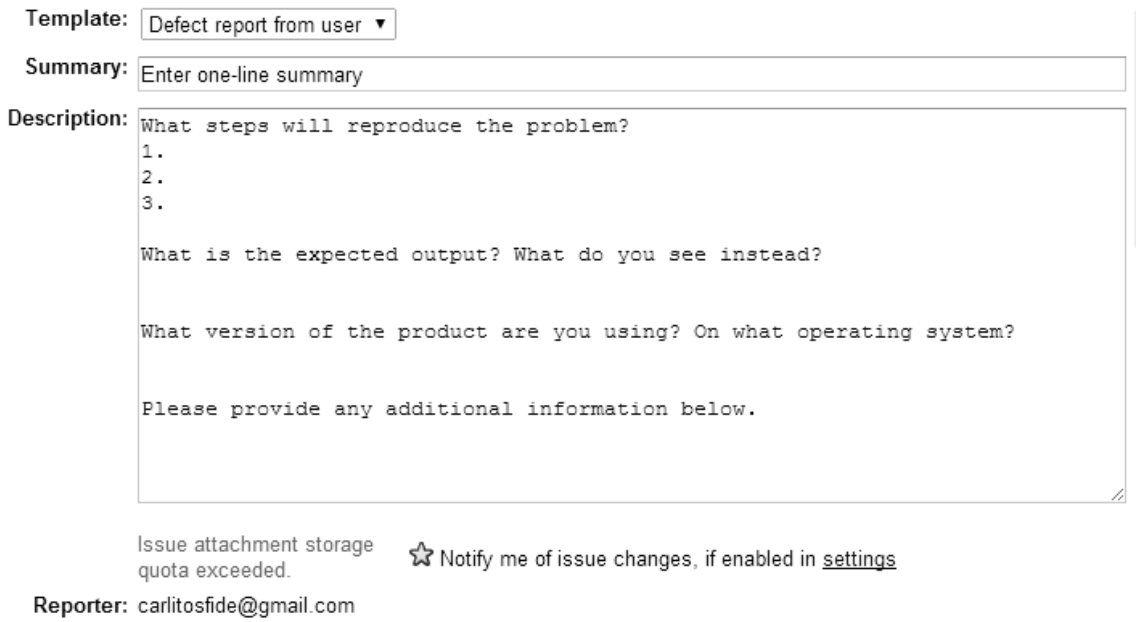


Figura G.8: Pantalla Creación de Defecto en Google Code

G.2.2. Eclipse

Figura G.9: Logo Eclipse

Eclipse (Figura G.9) es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (en inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Este fue el entorno que se utilizó para la implementación de la solución del error, ya que era el recomendado en la página. También se utilizó para la crear los parches que más adelante se enviaron por correo, Figura G.10.

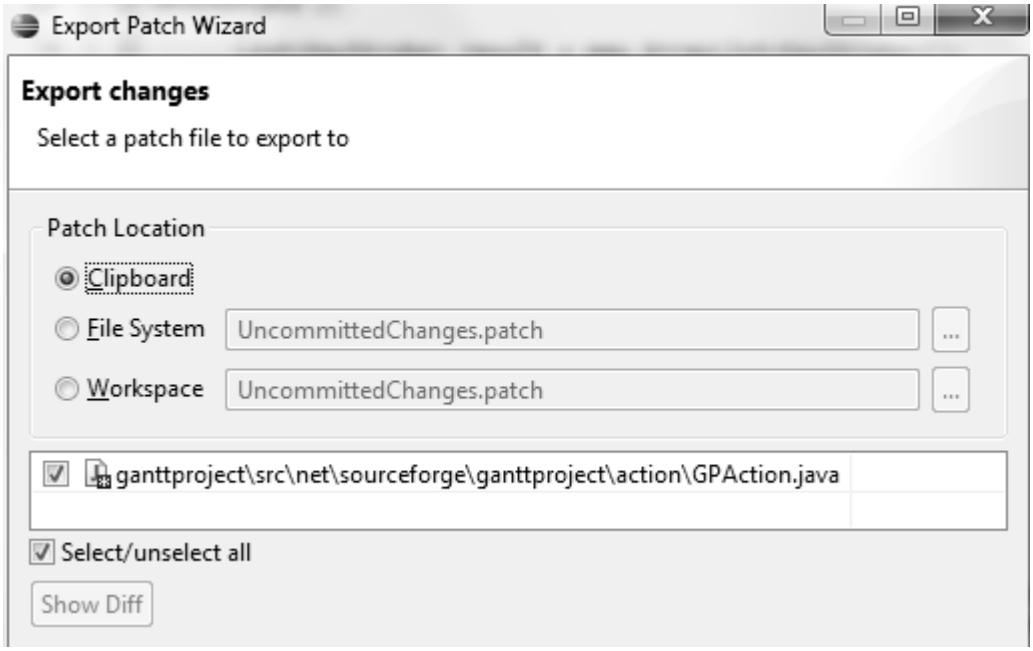


Figura G.10: Pantalla Creación de Parche en Eclipse

G.3. Comunicación con la Comunidad

G.3.1. Petición Asignación de Implementación

Original:

Hi all, i would be interested in correcting this error, if no problems i´m on it.

Castellano:

Hola a todos, estaría interesado en corregir este error, si no hay ningún problema me pongo a ello.

G3.2. Aceptación a la Petición de Corrección

Original:

Go ahead :)

Castellano:

Adelante :)

G.3.3. Correo Electrónico Solución del Error

Para: contribution@ganttproject.biz
De: carlitosfide@gmail.com

CC: ddarashev@gmail.com
Asunto: Parche
Adjunto: GPAction.patch

G.3.4. Correo Electrónico RE: Solución del Error

Para: carlos3lb@hotmail.com
De: ddarashev@gmail.com
CC:
Asunto: RE: Parche

Original:

Changes sent not only remove the information from the menus but other elements using GPAction. This option is not acceptable, sorry.

In the following link you will find information JMenuItems who can be useful http://docs.oracle.com/javase/6/docs/api/javax/swing/JComponent.html#setToolTipText (java.lang.String)

Castellano:

Los cambios enviados no solo eliminan la información de los menus sino de otros elementos que utilizan GPAction. Esta opción no es aceptable, lo siento.

En el siguiente enlace encontrarás información de JMenuItems que puede serte útil http://docs.oracle.com/javase/6/docs/api/javax/swing/JComponent.html#setToolTipText (java.lang.String).

G.3.5. Correo Electrónico RE2: Solución del Error

Para: carlos3lb@hotmail.com
De: ddarashev@gmail.com
CC:
Asunto: RE: Parche

Original:

Error solved. Other relevant files have been modified. Thank you for your cooperation

Castellano:

Error resuelto. Otros ficheros afectados han sido modificados. Gracias por la colaboración

G.4. Modificaciones en el Código

Debido al tamaño de los archivos modificados, a continuación se muestran solamente los cambios más relevantes realizados.

Se añade la función que se muestra en la Figura G.11, que crea un Tooltip con texto a null, lo que servirá para solucionar el error.

```
public static JMenu createTooltiplessJMenu(Action action) {
  JMenu result = new JMenu(action) {
    @Override
    public JMenuItem add(Action a) {
      JMenuItem result = super.add(a);
      result.setToolTipText(null);
      return result;
    }
  };
  result.setToolTipText(null);
  return result;
}
```

Figura G.11: Modificación Código UIUtil.java

En las partes del código donde se crea un menú que deseamos ocultar, se pasará de la Figura G.12 a la Figura G.13, con la llamada a la nueva función creada.

```
JMenu mServer = new JMenu(GPAction.createVoidAction("webServer"));
mServer.add(openURLAction);
mServer.add(saveURLAction);
add(mServer);
```

Figura G.12: Código Inicial ProjectMenu.java

```
JMenu mServer = UIUtil.createTooltiplessJMenu(GPAction.createVoidAction("webServer"));
mServer.add(openURLAction);
mServer.add(saveURLAction);
add(mServer);
```

Figura G.13: Modificación Código ProjectMenu.java

Anexo H. Bitácora de Colaboración en Oportunidad OpenHatch

H.1. Descripción

Figura H.1: Logo OpenHatch

OpenHatch (Figura H.1) es una página web en la que aparecen un gran número, del orden de 25.000, de oportunidades de colaboración con proyectos OSS. Para realizar la búsqueda de las colaboraciones, la página permite filtrar por proyecto, por número de errores y por lenguaje de programación, Figura H.2.

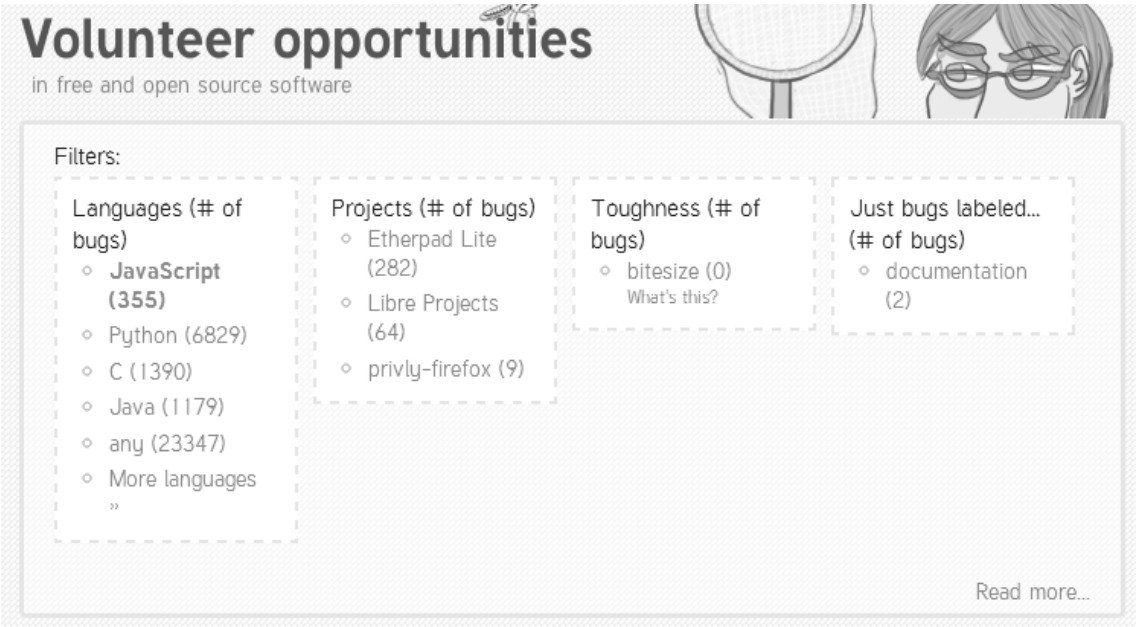


Figura H.2: Pantalla Logo OpenHatch

La colaboración realizada se encontró buscando por el lenguaje PHP y mirando las oportunidades existentes. Al pulsar en ellas se despliega información detallada sobre la funcionalidad a realizar y la forma de contactar con la comunidad, en este caso mediante correo electrónico.

La funcionalidad solicitada se basaba en, mediante llamadas a la API de Twitter, averiguar el grado de conexión entre dos usuarios de Twitter. Debido a no ser un proyecto finalizado, no existía página o sistema de tickets por lo que toda la comunicación se realizó por correo electrónico.

H.2. Herramientas

A continuación se describen las herramientas que se han utilizado durante la colaboración en este proyecto.

H.2.1. Twitter API

Figura H.3: Logo Twitter

Twiter (Figura H.3) es una conocida red social que incorpora una *Application Programming Interface* (API) abierta para todo tipo de desarrolladores, lo cual supone una gran ventaja para todos aquellos que quieran integrar Twitter como un servicio tanto en otras aplicaciones web como en aplicaciones de escritorio o móviles. Dicha API dispone de una consola, Figura H.4, para probar los métodos que ofrece la aplicación.

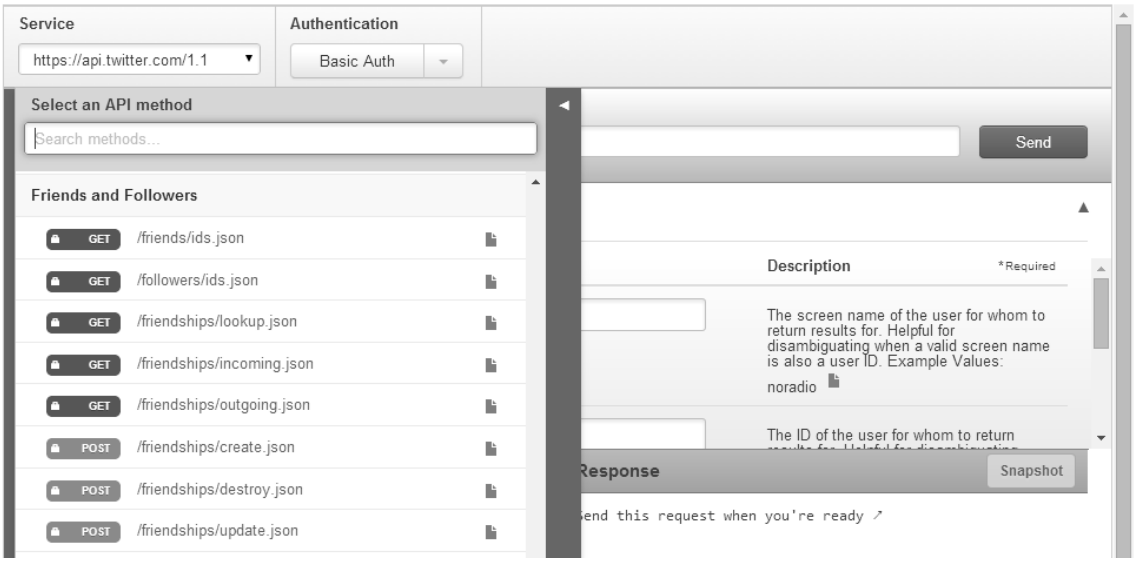


Figura H.4: Consola de Desarrollo API de Twitter

Esta consola se ha utilizado para probar las llamadas a la API que se nos indicaron desde la comunidad ya que se desconocía totalmente el funcionamiento.

H.2.2. Herramientas de Desarrollo

Debido a que se trataba de una serie de funciones en JavasScript, para realizar el código se ha utilizado simplemente el editor de NotePad++ y para realizar las pruebas Google

Chrome, que permite *debugear* el código y resulta muy útil a la hora de probar algoritmos.

H.3. Comunicación con la Comunidad

H.3.1. Correo Electrónico Solicitud Colaboración

Para: tim.msproject@gmail.com
De: carlos3lb@hotmail.com
CC:
Asunto: Colaboration

Original:

Hi,

I am a student of computer science concerned with the development of the functionality described in this task. I guarantee a good job, because I know very well the language used.

Thanks for your time, I hope your answer.

Carlos

Enviado desde Correo de Windows

Castellano:

Hola,

Soy un estudiante de Ingeniería informática interesado en realizar el desarrollo descrito en la oportunidad. Garantizo un buen trabajo ya que conozco muy bien el lenguaje utilizado.

Gracias por su tiempo, espero su respuesta.

Carlos

Enviado desde Correo de Windows

H.3.2. Correo Electrónico RE: Solicitud Colaboración

Para: carlos3lb@hotmail.com
De: tim.msproject@gmail.com
CC:

Asunto: RE: Colaboration

Original:

Hi Carlos,

Accept your request for collaboration. In the coming days I will send you the documentation needed for development.

Thanks for your interest

Tim

Castellano:

Hola Carlos,

Aceptamos tu petición de colaboración. En los próximos días te enviaré la documentación necesaria para el desarrollo.

Gracias por tu interés

Tim

H.3.3. Correo Electrónico Información sobre la Funcionalidad

Para: carlos3lb@hotmail.com
De: tim.msproject@gmail.com
CC:
Asunto: Implementation Information

Original:

Hi Carlos,

We are working on a web application for managing social networks. One of the features we want to add is that you have offered to develop.

Functionality:

The user will enter the screen_name (@ user) of two users, and the application will return a string of users.

This string is a minimum number of people doing retweets join the two users listed. The application also indicates the level (number of retweets) associated with this communication.

The development will take place in Javascript. There are PHP calls to retrieve and save the DDBB users. I am attaching a .zip with available functions.

The result should be painted in the calling html. To do this you have two div ("d1" and "d2") and you must indicate the level and a list of the users involved.

If you have questions feel free to ask.

Information will be obtained from the Twitter API. Later I will send more information.

Tim

Castellano:

Hola Carlos,

Estamos trabajando en una aplicación web para gestionar redes sociales. Una de las funcionalidades que queremos añadir es la que te has ofrecido a desarrollar.

Funcionalidad:

El usuario introducirá los screen_name (@ usuario) de los usuarios que desea conectar y la aplicación le devolverá una cadena de usuarios.

Esta cadena es una serie mínima de personas que haciendo retweets unirá a los dos usuarios indicados. La aplicación también indica el nivel (número de retweets) que conlleva esta comunicación.

El desarrollo se realizará en Javascript y podrá realizar llamadas PHP para recuperar y guardar usuarios en la DDBB. Te adjunto un .zip con las funciones disponibles.

El resultado deberá pintarse en el html que realiza la llamada. Para ello dispones de dos div ("d1" y "d2") y deberás indicar el nivel y una lista con los usuarios implicados.

Si tienes dudas, no dudes en preguntar.

La información de Twitter se obtendrá de su API. Más tarde te enviaré más información.

Tim

Documento Adjunto: Fichero comprimido con las funciones PHP

H.3.4. Correo Electrónico Información sobre la API de Twitter

Para: carlos3lb@hotmail.com
De: tim.msproject@gmail.com
CC:
Asunto: FW: Implementation Information

Original:

This is the link to the Twitter API:

https://dev.twitter.com/console

Note that there is a limit of API calls. On the other hand try to use the DDBB to avoid calls to Twitter, which is slower.

Before starting the development we ask you to send us an outline with calls to twitter and DDBB to assess the efficiency of development.

Tim

Castellano:

Éste es el enlace a la API de Twitter:

https://dev.twitter.com/console

Ten en cuenta que existe un límite de llamadas a la API. Por otro lado intenta utilizar la DDBB para evitar llamadas a Twitter, que es más lento.

Antes de empezar con el desarrollo te pido que nos envíes un esquema con las llamadas a Twitter y a la DDBB, para valorar la eficiencia del desarrollo.

Tim

Documento Adjunto: Documento con algunas de las llamadas a la API de Twitter

H.3.5. Correo Electrónico RE: Información sobre la API de Twitter

Para: tim.msproject@gmail.com
De: carlos3lb@hotmail.com

CC:
Asunto: RE :FW: Implementation Information

Original:

Hello,

Under the scheme calls to the Twitter API and DDBB.

When I bear the approval to begin development.

regards

Carlos

Enviado desde Correo de Windows

Castellano:

Hola,

Adjunto el esquema de llamadas a la API de Twitter y a la DDBB.

Cuando me deis el visto bueno comenzaré con el desarrollo.

Un saludo

Carlos

Enviado desde Correo de Windows

Documento Adjunto: Esquema de llamadas, Figura 5.1 y Figura 5.2

H.3.6. Correo Electrónico Ok para Iniciar la Implementación

Para: carlos3lb@hotmail.com
De: tim.msproject@gmail.com
CC:
Asunto: RE :FW: Implementation Information

Original:

Hello Carlos,

Good job, you can start with the implementation.

We recommend you make a html form to test a functionality.

If you need something write us.

Regards

Tim

Castellano:

Hola Carlos,

Buen trabajo, puedes empezar con la implementación.

Te recomiendo que te hagas un html con un form para probar la funcionalidad.

Si necesitas algo escríbenos.

Un saludo

Tim

H.3.7. Correo Electrónico Envío Código Completado

Para: tim.msproject@gmail.com
De: carlos3lb@hotmail.com
CC:
Asunto: Completed Code

Original:

Hello,

Under the code with the required functionality. If he had a problem or something to be changed, please tell me. If possible I would love to continue to contribute in this project.

Thank you very much,

Carlos

Enviado desde Correo de Windows

Castellano:

Adjunto el código con la funcionalidad requerida. Si hubiese algún problema o algo que haya que cambiar, por favor díganmelo.

Si fuese posible me encantaría poder seguir contribuyendo en este proyecto.

Muchas gracias,

Saludos.

Carlos

Enviado desde Correo de Windows

H.4. Código

A continuación se muestra el código de la aplicación. Se ha traducido totalmente y se han ocultado algunas funciones que no son relevantes para su entendimiento:

```
 /**********************VARIABLES GLOBALES**********************/
var follow = []; //almacena los que siguen
var followers = []; //almacena los seguidores
var resultadoFinal = []; //almacena el resultado final con los screen_name
var pag; //almacena la pagina para grabar followers y friends
var a;
var b;
var encontrado=0; //bandera elemento encontrado
var i;
var j;
var coincide=0; //bandera coincidencia elemento
//PILA ARBOL:
var fin=0; //fin pila
var inicio=0; //inicio pila
var pila=[]; //array que almacena la pila
//PILA VISITADOS:
var visitados=[];  //array visitados
var v=0; //proximo a visitar
var nivelArbol=0;//controla si se pide a twitter o a la BD
/*************************************************************/

/************************FUNCIONES****************************/
/*nodo:
Estructura para almacenar los nodos del arbol
usuario1 > ... > usuarioN */
function nodo(id, padre){
   this.id = id;
   this.padre = padre;
}

/*pintarNiveles:
muestra los niveles a partir del nivel 2*/
function pintarNiveles(visitados,visitado,follower,n,modo){
}

/*pintarNivel1:
Muestra el resultado en caso de ser nivel 1
usuario1 > usuario2 */
```

```
function pintarNivel1(u1, u2){
	var cad=[];
	cad[0]= u2;
	cad[1]= u1;
	convertirIDs(0, cad, 1, 2,0);
}

/*convertirIDs:
Almacena en el array resultadoFinal los nombres de
los usuarios que conforman los niveles.
Si no estan en la base de datos, los pide a
twitter y los inserta en la DB*/
function convertirIDs(indice, cadena, nivel, max,modo){
}

/*pintarResultado:
pinta el resultado final*/
function pintarResultado(nivel,modo){
}

/*pedirSeguidoresDe:
Almacena en la variable global followers los seguidores del
usuario indicado por id*/
function pedirSeguidoresDe(id,modo){
	var seguidores = [];
	$.ajax({
		url: "http://api.twitter.com/1.1/followers/ids.json?id="+id+"&callback=?",
		dataType: 'json',
		async: false,
		success: function(data) {
			str="+data.ids+";
			followers=str.split(",");
			siguienteSeguidoresDe(data.next_cursor_str,id,modo);
		}
	});
}

/*siguienteSeguidoresDe:
Incluye en la variable followers los seguidores del
id indicado mirando en las diferentes paginas si tiene
mas de 5000 seguidores*/
function siguienteSeguidoresDe(pagina,id,modo){
}

/*****************************FUNCIONES NIVEL BAJO*******************************/
/*pedirSeguidoresDeNivelBajo:
Elimina la lista de seguidores de la base de datos
(todas sus paginas)
y llama a otra funcion que los actualizara
*/
function pedirSeguidoresDeNivelBajo(iduser,modo){
	$.ajax({
		type: "POST",
	  url: "php/deleteUserFollowers.php",//peticion a la DB
	  data: {
			id: iduser
		},
	  datatype: 'json',
	  success: function(msg){
			pedirSeguidoresDeXX(iduser,modo);
		}
	});
}


/*pedirSeguidoresDeXX:
Almacena en la variable global followers los seguidores del
usuario indicado por id y actualiza la base de datos.
Si hay mas paginas llama a la funcion que las solicita*/
function pedirSeguidoresDeXX(iduser,modo){
}

/*siguienteSeguidoresDeXX:
Incluye en la variable followers los seguidores del
id indicado mirando en las diferentes paginas si tiene
```

```
mas de 5000 seguidores. Va creando paginas en la base de datos
con los ids de los seguidores y la pagina*/
function siguienteSeguidoresDeXX(pagina,iduser,modo){

}

/*pedirSeguidosPorNivelBajo:
Elimina la lista de seguidos de la base de datos
(todas sus paginas)
y llama a otra funcion que los actualizara
*/
function pedirSeguidosPorNivelBajo(iduser,modo){
	$.ajax({
		type: "POST",
	  url: "php/deleteUserFriends.php",//peticion a la DB
	  data: {
			id: iduser
		},
	  datatype: 'json',
	  success: function(msg){
			pedirSeguidosPorXX(iduser,modo);
		}
	});
}

/*pedirSeguidosPorXX:
Almacena en la variable global follow los seguidos por el
usuario indicado por id y actualiza la base de datos.
Si hay mas paginas llama a la funcion que las solicita*/
function pedirSeguidosPorXX(iduser,modo){
}

/*siguienteSeguidosPorXX:
Incluye en la variable follow los seguidos del
id indicado mirando en las diferentes paginas si tiene
mas de 5000 seguidores. Va creando paginas en la base de datos
con los ids de los seguidos y la pagina*/
function siguienteSeguidosPorXX(pagina,iduser,modo){

}
/**********************FUNCIONES NIVEL ALTO*****************************/
/*pedirSeguidosPorBD:
Pide a la BD los seguidos por un usuario, si no los tiene
llama a una funcion que se los pide a twitter
*/
function pedirSeguidosPorBD(iduser,modo){
	$.ajax({
		type: "POST",
	  url: "php/selectUserFriends.php",//peticion a la DB
	  data: {
			id: iduser
		},
	  datatype: 'json',
	  success: function(msg){
			var response = eval('(' + msg + ')');
			if(response.ok==1){
				str=response.friends;
				follow=str.split(",");//se insertan en el array
				if(modo==0){
					sigue();
				}else{
					sigue2B();
				}
			}else{
				pedirSeguidosPorXX(iduser,modo);//si no estan en la BD, aki se piden a twitter
			}
		}
	});
}

/*pedirSeguidoresDeBD:
Pide a la BD los seguidores de por un usuario, si no los tiene
llama a una funcion que se los pide a twitter
*/
function pedirSeguidoresDeBD(iduser,modo){
```

```
	$.ajax({
		type: "POST",
	  url: "php/selectUserFollowers.php",//peticion a la DB
	  data: {
			id: iduser
		},
	  datatype: 'json',
	  success: function(msg){
			var response = eval('(' + msg + ')');
			if(response.ok==1){
				str=response.followers;
				followers=str.split(",");//se insertan en el array
				if (modo==0){
					sigue2();
				}else{
					sigueB();
				}
			}else{
				pedirSeguidoresDeXX(iduser,modo);//si no estan en la BD, aki se piden a twitter
			}
		}
	});
}


/**********************************************************************/


/*ini:
funcion que inicia el proceso. Realiza una peticion a la base de datos,
si encuentra el id lo usa, en caso contrario llama a la funcion
initwitter*/
function ini(NombreA,NombreB){
			$.ajax({
				type: "POST",
			  url: "php/selectUser.php",
			  data: {
					name: NombreA
				},
			  datatype: 'json',
			  success: function(msg){
					var response = eval('(' + msg + ')');
					if(response.ok==1){
						a = response.id;
						SeguidoresA=response.followers_count;
						ini2(NombreB); //ok, al segundo usuario
					}else{
						iniTwitter(NombreA,NombreB); //err, se pide a twitter
					}
			  }
			});
}

/*iniTwitter:
Realiza la peticion del usuario A a twitter y guarda el usuario en la
base de datos.
*/
function iniTwitter(NombreA,NombreB){
}

/*ini2:
funcion que solicita los datos del usuario b. Realiza una peticion a la base de datos,
si encuentra el id lo usa, en caso contrario llama a la funcion
ini2twitter*/
function ini2(NombreB){
	$.ajax({
		type: "POST",
	  url: "php/selectUser.php",
	  data: {
			name: NombreB
		},
	  datatype: 'json',
	  success: function(msg){
			var response = eval('(' + msg + ')');
			if(response.ok==1){
```

```
                                                b = response.id;
                                                if(SeguidoresA <= msg.friends_count){  //si a A le siguen menos que B sigue,

                        //expando A que es mas peque–o
                                                        iniAB();
                                                }else{
                                                        iniBA();
                                                }
                                        }else{
                                                ini2Twitter(NombreB); //peticion a twitter
                                        }
                                }
                        });
}

/*ini2Twitter:
Realiza la peticion del usuario B a twitter y guarda el usuario en la
base de datos.
*/
function ini2Twitter(NombreB){
}

//PROCESO A ----> B

/*iniAB:
funcion que inicia el proceso de A hacia B*/
function iniAB(){
        pedirSeguidosPorNivelBajo(b,0);
}

/*sigue:
Continua con la ejecucion tras pedir los usuarios a los que sigue b*/
function sigue(){
        for(j=0;j<follow.length;j++){//buscamos a en follow
                if (follow[j]==a){
                        encontrado=1;
                        pintarNivel1(a, b);//si encontrado pinta el resultado
                }
        }
        pila[fin]=new nodo(a,"ax");//nodo raiz en la pila
        fin++;
        whle();
}

/*whle:
Comprueba si el nodo actual ya ha sido visitado,
en ese caso avanza la pila*/
function whle(){
        if (encontrado==1){
                return true;
        }else{
                coincide=0;
                for(i=0;i<visitados.length;i++){ //comprueba si ya ha sido expandido el tope de la pila
                        if (visitados[i]==pila[inicio].id){
                                inicio ++;
                                coincide=1;
                                break;
                        }
                }
                if (coincide==0){ //si no ha sido expandido lo expande
                        if (nivelArbol==0){
                                pedirSeguidoresDeNivelBajo(pila[inicio].id,1);
                                nivelArbol++;
                        }else {
                                pedirSeguidoresDeBD(pila[inicio].id,0);
                        }
                }
        }
}

/*sigue:
Continua con la ejecucion tras pedir los seguidores de alguien*/
function sigue2(){
        visitados[v]=new nodo(pila[inicio].id, pila[inicio].padre); //inserta el ultimo expandido en visitados
        inicio ++;
```

```
		v++
		for(i=0;i<followers.length;i++){ //busca coincidencias
			for(j=0;j<follow.length;j++){
				if (followers[i]==follow[j]){
					pintarNiveles(visitados,visitados[v-1],followers[i],b,0); //si ok pinta el resultado
					encontrado=1;
				}
			}
			if(encontrado==1){
				return true;
			}else{
				pila[fin]=new nodo(followers[i],visitados[v-1].id); //si err lo inserta en la pila
				fin ++;
			}
		}
		whle();
}

//PROCESO B ----> A

/*iniBA:
funcion que inicia el proceso de B hacia A*/
function iniBA(){
		pedirSeguidoresDeNivelBajo(a,1);
}

/*sigueB:
Continua con la ejecucion tras pedir los usuarios a los que sigue b*/
function sigueB(){
		for(j=0;j<followers.length;j++){//buscamos b en followers
			if (followers[j]==b){
				encontrado=1;
				pintarNivel1(a, b);//si encontrado pinta el resultado
			}
		}
		pila[fin]=new nodo(b,"ax");//nodo raiz en la pila
		fin++;
		whleB();
}

/*whleB:
Comprueba si el nodo actual ya ha sido visitado,
en ese caso avanza la pila*/
function whleB(){
		if (encontrado==1){
			return true;
		}else{
			coincide=0;
			for(i=0;i<visitados.length;i++){//comprueba si ya ha sido expandido el tope de la pila
				if (visitados[i]==pila[inicio].id){
					inicio ++;
					coincide=1;
					break;
				}
			}
			if (coincide==0){
				if (nivelArbol==0){//si no ha sido expandido lo expande
					pedirSeguidosPorNivelBajo(pila[inicio].id,1);
					nivelArbol++;
				}else {
					pedirSeguidosPorBD(pila[inicio].id,1);
				}
			}
		}
}

/*sigue2B:
Continua con la ejecucion tras pedir los seguidos por alguien*/
function sigue2B(){
		visitados[v]=new nodo(pila[inicio].id, pila[inicio].padre);//inserta el ultimo expandido en visitados
		inicio ++;
		v++
		for(i=0;i<follow.length;i++){//busca coincidencias
			for(j=0;j<followers.length;j++){
				if (follow[i]==followers[j]){
```

```
                                        pintarNiveles(visitados,visitados[v-1],follow[i],a,1);//si ok pinta el resultado
                                        encontrado=1;
                                }
                        }
                        if(encontrado==1){
                                return true;
                        }else{
                                pila[fin]=new nodo(follow[i],visitados[v-1].id);//si err lo inserta en la pila
                                fin ++;
                        }
                }
                whleB();
}
```